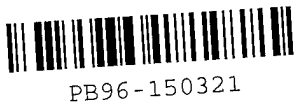
October 1989

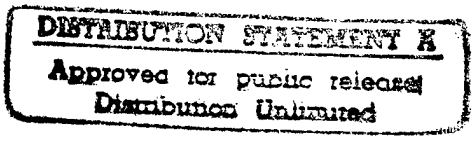
Report No. STAN-CS-89-1285



Numerical Potential Field Techniques for Robot Path Planning

by

Jérôme Barraquand, Bruno Langlois, and Jean-Claude Latombe



Department of Computer Science

Stanford University

Stanford, California 94305

19970630 073

DTIC QUALITY INSPECTED 1

unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-89-1285	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Stanford University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Stanford, CA 94305	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA/ONR 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAA21-89-C0002 / N00014-88-K-0620
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22161	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Numerical Potential Field Techniques for Robot Path Planning

12. PERSONAL AUTHOR(S)
Jerome Barraquand, Bruno Langlois, Jean-Claude Latombe

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
research	FROM _____ TO _____		38

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

See following page.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Jean-Claude Latombe	22b TELEPHONE (Include Area Code) 415-723-0350 / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

Numerical Potential Field Techniques for Robot Path Planning

Jérôme Barraquand, Bruno Langlois and Jean-Claude Latombe
Robotics Laboratory
Computer Science Department, Stanford University
Stanford, CA 94305, USA

Abstract: We propose a new approach to robot path planning which consists of incrementally building a graph connecting the local minima of a potential function defined in the robot's configuration space and concurrently searching this graph until a goal configuration is attained. Unlike the so-called "global" path planning methods, this approach does not require an expensive computation step before the search for a path can actually start. On the other hand, it searches a graph that is usually much smaller than the graph searched by the so-called "local" methods. We describe a collection of specific techniques that allow to engineer various implementations of our path planning approach. The purpose of these techniques is to (1) construct "good" potential fields and (2) efficiently escape their local minima (i.e., efficiently build the local minima graph). They are based on the use of multiscale pyramids of bitmap arrays for representing both the robot's workspace and configuration space. This distributed representation makes it possible to construct potential fields *numerically*, rather than *analytically*, in relation to other efficient numerical techniques. We have implemented these techniques in a path planner, which turns out to be both very fast and capable of handling many degrees of freedom. The planner has solved a variety of problems. Some of them are far beyond the capabilities of previously existing planners.

Acknowledgements: This research was funded by DARPA contract DAAA21-89-C0002 (Army), DARPA contract N00014-88-K-0620 (Office of Naval Research), CIFE (Center for Integrated Facility Engineering), CIS (Center of Integrated Systems), and DEC (Digital Equipment Corporation).

1 Introduction

In this paper we describe several *numerical potential field techniques* for robot motion planning. We have implemented these techniques in a path planner which turns out to be both very fast and capable of handling many degrees of freedom (DOF's). In particular, the planner has been able to plan the motions of mobile robots with 3 DOF's (two translations and one rotation) two orders of magnitude faster than most other existing planners. It has also generated paths of robots with many DOF's in reasonable amount of time. For example, difficult paths for a non-serial manipulator with 10 joints (some revolute, some

prismatic) were produced in 1 to 5 minutes[1]. Paths were also generated in a 3D workspace for a manipulator with 31 DOF's in about 15 minutes of computation time. These results are far beyond the capabilities of previously implemented planners.

The problem of generating collision-free paths has attracted considerable interest during the past years [Schwartz, Hopcroft and Sharir, 1987] [Schwartz and Yap, 1987]. By simplifying, we can say that two extreme approaches have been explored, the "global" one and the "local" one. The global approach consists of first constructing a concise representation of the connectivity of the set of collision-free configurations of the robot in the form of a "connectivity graph" and then searching this graph for a path. Various techniques have been devised, e.g.: exact cell decomposition [Schwartz and Sharir, 1983], approximate cell decomposition [Brooks and Lozano-Pérez, 1983] [Faverjon, 1986], retraction on a curve network [Ó'Dúnlaing, Sharir and Yap, 1983]. The local approach consists of searching a grid placed onto the robot's configuration space [Donald, 1984]. Heuristics computed from partial information about the geometry of the configuration space are used to guide the search (and eventually to construct the grid itself). A widely used heuristic technique guides the search for a path along the flow of the negated gradient vector field generated by an artificial potential field [Khatib, 1986] [Faverjon and Tournassoud, 1987].

The drawback of the global approach is that it requires an expensive precomputation step – the construction of the connectivity graph – before the search for a path can actually start. Since the time required by the computation of the connectivity graph is typically exponential in the dimension n of the configuration space (i.e., the number of DOF's), the approach is untractable even for reasonably small values of n. To our knowledge, no effective planner has been implemented using this approach with $n > 4$. Instead, the local approach requires no expensive precomputation step before starting the search of a path. Consequently, in favorable cases, it runs substantially faster than any global method. But, since the search graph (i.e., the grid) is considerably larger than the connectivity graph searched by global methods, it may require much more time than global methods in unfavorable cases. In order to deal with this difficulty, local methods need powerful heuristics to guide the search. But known such heuristics have the drawback of eventually leading the search to dead-ends, for instance local minima of the potential field, which may be quite difficult to escape. One may think of constructing a potential field with no other local minimum than the goal configuration (in the connected subset of the free space containing the goal configuration), but the analytical definition of such a potential turns out to be very difficult (e.g., see [Koditschek and Rimon, 1989]). Futhermore, even if a definition was available, it is likely that its computation would constitute an expensive precomputation step before path generation, similar in drawback to the construction of a connectivity graph.

There are many reasons motivating the development of a fast path planner capable of dealing with many DOF's. For instance, a (semi-)autonomous robot will have to generate its paths on-line, based on its current model of the world, and to react rapidly to contingencies. Although one may envision a robot that learns about its workspace and memorizes a variety

[1] Most of the experiments reported in this paper were carried out on a DEC 3100 MIPS-based workstation using simulated robots.

of typical paths, it is by far more appropriate to have a fast "real-time" path planner[2]. Robots will typically combine one or several arms mounted on a mobile vehicle; for instance, space robots may consist of several manipulator arms attached to a free-flying platform [Schneider, 1989]. Planning the paths of such robots will require to be able to handle many DOF's, specially if cooperation among them is needed. One may argue that, most of the time, at every instant, the planner has to worry only about a subset of these DOF's. But determining which DOF's are important at every instant is also part of planning and hence should be handled by the planner. Fast path planning capabilities may also be extremely useful for off-line programming of industrial and construction robots, which requires quick man-machine graphical interaction.

We propose a new approach to path planning that attempts to combine the advantages of both the local approach (avoid expensive precomputation) and the global approach (search a concise graph). This approach consists of incrementally building a graph connecting the local minima of a potential function defined in the configuration space of the robot and concurrently searching this graph until a goal configuration is attained. The graph of local minima plays a role similar to that of the connectivity graph in the global approach. The major difference, however, is that it is constructed in an incremental fashion during the search. Hence, our approach does not require an expensive precomputation step, although it definitely searches a much smaller graph than the discretization grid thrown over configuration space. In this paper, we describe a collection of specific techniques that allow to engineer various implementations of this path planning approach[3]. The purpose of these techniques is to (1) construct "good" potential fields and (2) efficiently escape their local minima (i.e., efficiently build the local minima graph). They are based on the use of multi-scale pyramids of bitmap arrays for representing both the workspace and the configuration space of the robot. These representations allow us to construct potential fields *numerically*, rather than *analytically*, in relation to other efficient numerical techniques (e.g., collision checking, valley tracking).

In Section 2, we describe the hierarchical bitmap representations of the workspace and the configuration space of a robot. In Section 3, we propose several ways of constructing numerical potential fields in the robot's configuration space. In Sections 4 through 7, we present four path planning techniques – respectively called "best-first motion", "random motion", "valley-guided motion" and "constrained motion" techniques – which are based on different ways of escaping the local minima of these potential fields. We have implemented all these techniques and, for each one, we give experimental results. Each technique admits many straightforward variants and, in the course of our experimentation, we ran several variants successfully. Hence, within some limits, the algorithms presented in this paper may be adapted so that they better fit the characteristics of a specific application domain and computing system.

[2]To that respect, it is interesting to remember that not so many years ago, it was proposed to compute the inverse kinematics of a manipulator by storing the numerical values of the inverse Jacobian matrix of the forward kinematic map at many configurations of the manipulator. Today, the computation of the inverse kinematics is routinely done in real-time without having to store inverse Jacobian matrices.

[3]Some of these techniques were previously presented in [Barraquand, Langlois and Latombe, 1989] and [Barraquand and Latombe, 1989].

2 Bitmap Representations

2.1 Basic Terminology and Notations

We denote by $\mathcal{A}$ the robot and $\mathcal{W}$ its workspace. A standard Cartesian coordinate frame, denoted by $\mathcal{F}_\mathcal{W}$, is embedded in $\mathcal{W}$.

A **configuration** of $\mathcal{A}$ is a specification of the position of every points in $\mathcal{A}$ with respect to $\mathcal{F}_\mathcal{W}$. The **configuration space** of $\mathcal{A}$ is the space, denoted by $\mathcal{C}$, of all the possible configurations of $\mathcal{A}$. The subset of $\mathcal{W}$ occupied by $\mathcal{A}$ at configuration $\mathbf{q}$ is denoted by $\mathcal{A}(\mathbf{q})$.

The workspace contains a finite number of obstacles denoted by $\mathcal{B}_i$, with $i = 1, ..., r$. We denote by $\mathcal{W}_{empty}$ the region $\mathcal{W} - \bigcup_{i=1}^{r} \mathcal{B}_i$. Each obstacle $\mathcal{B}_i$ maps in $\mathcal{C}$ to the set $\mathcal{CB}_i$ of configurations, called **C-obstacle**, where the robot and the obstacles intersect, i.e.:

$$\mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C} \,/\, \mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i \neq \emptyset\}.$$

The subset of configurations where the robot and the obstacles have no intersection, i.e.:

$$\mathcal{C}_{free} = \mathcal{C} - \bigcup_{i=1}^{r} \mathcal{CB}_i$$

is called the **free space**. A **collision-free path** (more simply, a **free path**) between two configurations $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ is any continuous map $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$, such that $\tau(0) = \mathbf{q}_{init}$ and $\tau(1) = \mathbf{q}_{goal}$.

The configuration space $\mathcal{C}$ is a nD manifold [Arnold, 1978]. For example, most mobile robots can be modelled as a 2D object $\mathcal{A}$ that can both translate and rotate in the plane. Then, $\mathcal{C}$ is the 3D manifold $\mathbf{R}^2 \times S^1$, where S^1 is the unit circle. In the case of a manipulator arm with n revolute joints, $\mathcal{C}$ is the nD manifold $(S^1)^n$, or a subset of that space when the motion of each joint is limited by mechanical stops.

Throughout this paper, we represent a configuration $\mathbf{q}$ of $\mathcal{A}$ by a list of n parameters, $(q_1, ..., q_n)$, where n is the dimension of $\mathcal{C}$. This parameterization is eventually augmented with modular arithmetic for some of the angular parameters. Hence, we represent $\mathcal{C}$ as a nD Cartesian space. For example, $\mathcal{C} = \mathbf{R}^2 \times S^1$ may be represented as $\mathbf{R}^2 \times \mathbf{R}/2\pi Z$. Any configuration $\mathbf{q}$ is then parameterized by (x, y, θ), where $(x, y) \in \mathbf{R}^2$ and $\theta \in [0, 2\pi)$ with modulo 2π arithmetic. Similarly, $\mathcal{C} = (S^1)^n$ may be represented as $(\mathbf{R}/2\pi Z)^n$. Any configuration is parameterized by $(q_1, ..., q_n)$ with $q_i \in [0, 2\pi)$, for every $i \in [1, n]$, and modulo 2π arithmetic on every q_i.

For each point $p \in \mathcal{A}$, one can consider the geometrical application that maps any configuration $\mathbf{q} = (q_1, ..., q_n) \in \mathcal{C}$ to the position $\mathbf{x} \in \mathcal{W}$ of p in the workspace. This map:

$$\begin{array}{rccc} X & : & \mathcal{A} \times \mathcal{C} & \rightarrow & \mathcal{W} \\ & & (p, \mathbf{q}) & \mapsto & \mathbf{x} = X(p, \mathbf{q}) \end{array}$$

is called **forward kinematic map**.

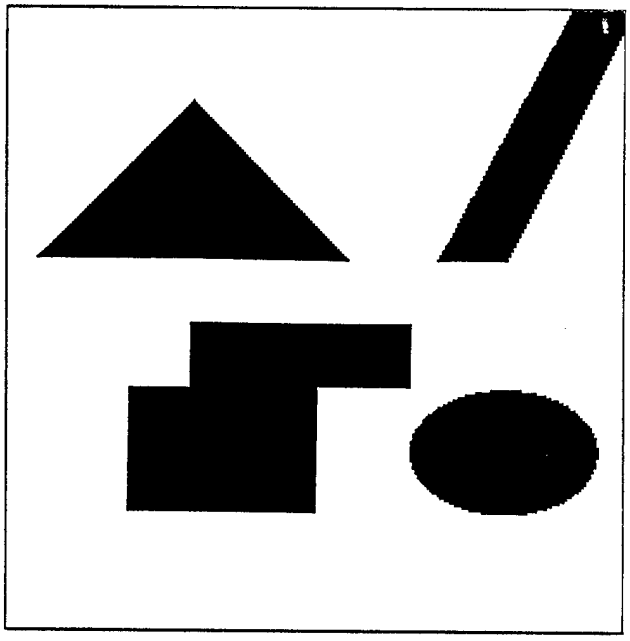


Figure 1: 256^2 Bitmap Description of a Workspace

2.2 Workspace Bitmap

In the following, we make the reasonable assumption that $\mathcal{W}$ is a bounded subset of $\mathbf{R}^d$, with $d = 2$ or 3. $\mathcal{W}$ is modelled as a multiscale pyramid of dD bitmap arrays. At each resolution level, the array is represented by a function:

$$\begin{array}{rccc} BM & : & \mathcal{W} & \rightarrow & \{1,0\} \\ & & \mathbf{x} & \mapsto & BM(\mathbf{x}) \end{array}$$

in such a way that the subset of points $\mathbf{x}$ such that $BM(\mathbf{x}) = 1$ represents the workspace obstacles and the subset of points $\mathbf{x}$ such that $BM(\mathbf{x}) = 0$ represents the empty part of the workspace, i.e. $\mathcal{W}_{empty}$.

The distance between the centers of two consecutive cells in the same line or the same column of an array is constant. Hence, at any level of resolution, the midpoints of the cells of the bitmap array form a regular grid denoted by $\mathcal{GW}$. The subset of the grid where BM evaluates to 0 is denoted by $\mathcal{GW}_{empty}$. For any integer $k \in [1, r]$, the k-heighborhood of a point $\mathbf{x}$ in a grid of dimension r is defined as the set of points in the grid having at most k coordinates differing from those of $\mathbf{x}$, the amount of the difference (if any) along any axis being the discretization step along that axis. In $\mathcal{GW}$, we always use the 1-neighborhood, except noticed otherwise (e.g., to track a curve or a surface). In a 2D workspace grid, this means that each point $\mathbf{x} = (i, j) \in \mathcal{GW}$ has a maximum of 4 neighbors: $\{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$. In a 3D workspace grid, each point has up to 6 neighbors.

Most of the experiments reported in this paper were carried out in a 2D workspace ($d = 2$). In those experiments, the coarsest level of resolution in the pyramid was 16^2 and the finest

1. For every $\mathbf{x} \in \mathcal{GW}_{empty}$, set $d_1(\mathbf{x})$ to infinity (i.e., a large number M).

2. Scan $\mathcal{GW}$ and identify every point $\mathbf{x}$ such that $BM(\mathbf{x}) = 1$ and one of its neighbors is in $\mathcal{GW}_{empty}$. Set L_0 to the list of these points. Include the points forming the frame boundary of $\mathcal{GW}$ in L_0. For every point $\mathbf{x}$ in L_0, set $d_1(\mathbf{x})$ to 0. Set i to 0.

3. For $i = 0, 1, 2, ...,$ while L_i until L_i is empty, do: initialize L_{i+1} to the empty list; for every point $\mathbf{x}$ in L_i, for every neighbor $\mathbf{y}$ of $\mathbf{x}$ in $\mathcal{GW}_{empty}$, if $d_1(\mathbf{y}) = M$ then set $d_1(\mathbf{y})$ to $i + 1$ and insert $\mathbf{y}$ at the end of L_{i+1}.

Figure 2: Algorithm Computing the d_1 Map

512^2. The bitmap representation of a particular workspace at the 256^2 resolution is shown in Figure 1 (1 = black; 0 = white). The workspace representation is given to the planner at the finest level of resolution. The other levels are automatically derived from it in a conservative fashion, so that $\{\mathbf{x} \in \mathcal{W} \ / \ BM(\mathbf{x}) = 0\} \subseteq \mathcal{W}_{empty}$. The scaling factor between two successive levels of resolution is 2, but a different factor could have been chosen.

In preparation to other algorithms to be described later, the planner computes the discretized L^1 distance $d_1(\mathbf{x})$ from every point $\mathbf{x} \in \mathcal{GW}_{empty}$ to the obstacles. This computation is performed according to the following "wavefront expansion" algorithm[4]. First, the points in the boundary of the obstacles are identified and the value of d_1 at these points is set to zero (the points in the contour of the bitmap are also included as boundary points). Next, the value of d_1 is set to 1 at all the neighbors of the boundary points in $\mathcal{GW}_{empty}$; to 2 at the neighbors of these new points (if not yet computed); etc ... The algorithm terminates when all $\mathcal{GW}_{empty}$ has been explored. A more formal description of the algorithm is given in Figure 2.

The time complexity of this algorithm is linear in the number of points in the grid $\mathcal{GW}$. Figure 3 displays contours of d_1 for the workspace of Figure 1.

2.3 Configuration Space Bitmap

Since we discretize the workspace in a hierarchical fashion, it is consistent to discretize the configuration space $\mathcal{C}$ as well, by constructing another multiresolution grid pyramid. This pyramid has as many levels of resolution as the workspace pyramid and the resolutions at each level of the two pyramids are tightly related, as developed below. At each level of resolution, we denote by $\mathcal{GC}$ the grid representing the configuration space and by $\mathcal{GC}_{free}$ the subset of the grid lying in the free space $\mathcal{C}_{free}$.

Let us denote by δ the distance between two adjacent points in a workspace grid $\mathcal{GW}$. In the workspace pyramid, δ varies between δ_{min} and δ_{max}. For example, let us assume that we have a workspace represented by a pyramid of arrays whose sizes are ranging between

[4]In this algorithm we normalize the distance between two neighbors in the grid to 1.

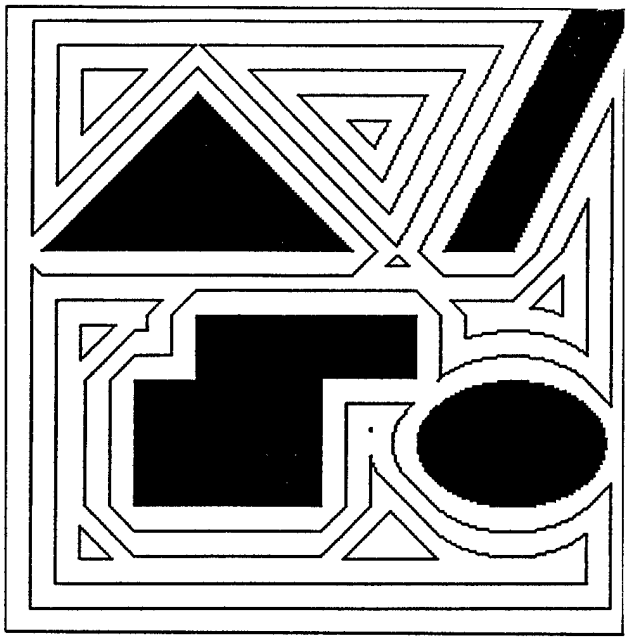


Figure 3: Contours of d_1 in a Workspace

16^2 and 512^2. If the distance is measured in percentage of the workspace diameter, we have $\delta_{min} = 1/512$ and $\delta_{max} = 1/16$.

By definition, the resolution of a grid is the logarithm of the inverse of the distance between two discretization points, in the base defined by the scaling factor between two successive resolution levels (2 in our implementation). In our example, the resolution r hence varies between $r_{min} = -\log_2(\delta_{max}) = 4$ and $r_{max} = -\log_2(\delta_{min}) = 9$.

Remember that we represent the configuration space $\mathcal{C}$ as a subset of a nD Cartesian space with $\mathbf{q} = (q_1, ..., q_n)$. For any given workspace resolution, say $r = -\log_2(\delta)$, the corresponding resolution $R_i = -\log_2(\Delta_i)$ of the discretization along the q_i axis of $\mathcal{C}$ is chosen in such a way that a modification of q_i by $\Delta_i = 2^{-R_i}$ generates a "small motion" of the robot in the workspace. By "small motion", we mean that any point $p \in \mathcal{A}$ moves by less than $nbtol \times \delta$, where $nbtol$ is a small number (typically, 1 or 2).

The relation between positions of robot points in the workspace and robot configurations is given by the forward kinematic map $X(p, \mathbf{q})$. For every point $p \in \mathcal{A}$, a modification of q_i by Δ_i results in a modification of each coordinate x_j of p ($j \in [1, d]$) by:

$$\frac{\partial x_j}{\partial q_i}(p, \mathbf{q})\Delta_i.$$

If we impose all workspace motions to be less than $nbtol \times \delta$, we must have:

$$\Delta_i = nbtol \times \delta / \sup_{p \in \mathcal{A}, \mathbf{q} \in \mathcal{C}, j \in [1,d]} \left(\frac{\partial x_j}{\partial q_i}(p, \mathbf{q}) \right) = nbtol \times \delta / J^i_{\sup}.$$

For a given robot, the numbers J^i_{sup} are generally straightforward to compute. This leads to compute the resolution R_i as:

$$R_i = r + \log_2(J^i_{\text{sup}}) - \log_2(nbtol)$$

As an example, let us consider a bar of length L moving freely in a 2D workspace. We can represent a configuration of the bar by (x_G, y_G, θ), where x_G and y_G are the coordinates of the center of gravity and θ is the orientation of the bar. Let us normalize x_G, y_G, and θ so that their values range between 0 and 1. We have

$$J^{x_G}_{\text{sup}} = J^{y_G}_{\text{sup}} = 1$$

and:

$$J^{\theta}_{\text{sup}} = \pi L.$$

If we set $nbtol = 2$, we get:

$$R_{x_G} = R_{y_G} = r - 1$$

and:

$$R_\theta = r + \log_2(\pi L) - 1.$$

This means that we need $2^1 = 2$ times less samples for x_G and y_G than for the workspace representation at each level of resolution, and $2/(\pi L)$ less samples for θ.

The planning techniques described below do not construct and/or use a complete representation of the configuration space grids, nor of the C-obstacles, since this would be too time and space consumming. Indeed, while the configuration space grids implicitly define the search space of the planner, in most practical cases, only a very small subset of the grids will be explored. It is clear, however, that in the worst case the grids would have to be exhaustively explored, requiring exponential time computation as any other existing path planning method.

3 Potential Field Construction

3.1 Overview

We describe below several ways of constructing numerical potential fields in the configuration space of a robot. In all cases, the construction consists of two major computing steps. First, potential fields, called **W-potentials**, are computed in the robot workspace $\mathcal{W}$. Each W-potential applies to a selected point in the robot, called **control point**, and pulls this point toward its goal position. The W-potentials at the various control points are then combined into another function, called **C-potential**, which is defined over the robot's configuration space.

More formally, let p_i, $i = 1, ..., s$, denote the control points in the robot. Each W-potential is a function:

$$\mathbf{U}_{p_i} : \mathbf{x} \in \mathcal{W}_{empty} \mapsto \mathbf{U}_{p_i}(\mathbf{x}) \in \mathbf{R}.$$

The C-potential is defined as:

$$\mathbf{U}(\mathbf{q}) = G(\mathbf{U}_{p_1}(X(p_1, \mathbf{q})), \ldots, \mathbf{U}_{p_s}(X(p_s, \mathbf{q})))$$

1. For every $\mathbf{x} \in \mathcal{GW}_{empty}$, set $\mathbf{U}_p(\mathbf{x})$ to infinity (i.e., a large number M).

2. Set $\mathbf{U}_p(\mathbf{x}_{goal})$ to 0, the list L_0 to $(\mathbf{x}_{goal})$, and the counter i to 0.

3. For $i = 0, 1, ...$, until L_i is empty, do: initialize L_{i+1} to the empty list; for every point $\mathbf{x}$ in L_i, for every neighbor $\mathbf{y}$ of $\mathbf{x}$ in $\mathcal{GW}_{empty}$, if $\mathbf{U}_p(\mathbf{y}) = M$ then set $\mathbf{U}_p(\mathbf{y})$ to $i+1$ and insert $\mathbf{y}$ at the end of L_{i+1}.

Figure 4: Algorithm Computing the Simple W-Potential

where G is called the **arbitration** function.

The rationale behind this two-step computation is to use the configuration space as a source of inspiration for constructing a "good" C-potential. As a matter of fact, the W-potentials computed by the algorithms of the next subsection are free of local minima. This allows us to construct C-potentials which avoid the robot to get trapped in simple concavities formed by the obstacles. However, since several W-potentials are typically combined in order to construct a C-potential, the combination may still have spurious local minima. These minima result from the fact that the various control points, which are related by fixed or variable kinematic constraints, are concurrently attracted toward their respective goal positions. Hence, they are competing among themselves to attain their respective goal positions. It is this competition which may create unwanted minima. The role of the function G is to arbitrate in this competition, hence the name of the function. Various arbitration functions are possible, resulting in more or less numerous, more or less deep local minima (Subsection 3.3).

3.2 Computation of W-Potentials

We propose two techniques for computing local-minima-free W-potentials. Other similar techniques can easily be developed. In the rest of the paper, we will refer to the two W-potentials defined below as the "simple W-potential" and the "improved W-potential", respectively.

3.2.1 Simple W-Potential

Let p be a control point in the robot and $\mathbf{U}_p(\mathbf{x})$ the corresponding W-potential. We want $\mathbf{U}_p$ to be free of local minima, i.e. to have a single minimum at the goal position of p in the connected subset of $\mathcal{W}_{free}$ containing this goal position. Indeed, as mentioned above, we think that this is a major heuristic step toward the construction of a C-potential with few or small spurious local minima.

Let $\mathbf{x}_{goal}$ be the goal position of p in $\mathcal{W}$. The simple W-potential[5] $\mathbf{U}_p$ is computed as follows. First, the value of $\mathbf{U}_p$ is set to 0 at $\mathbf{x}_{goal}$. Next, the value of $\mathbf{U}_p$ is set to 1 at the neighbors of $\mathbf{x}_{goal}$ in $\mathcal{G}_{empty}$; to 2 at the neighbors of these new points (if not yet computed), etc... The

[5] A similar potential has previously been proposed in [Jarvis and Byrne, 1987].

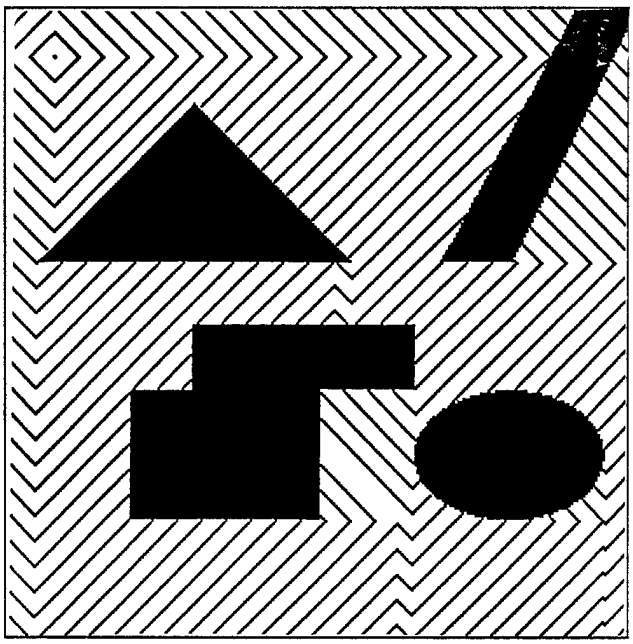


Figure 5: Contours of the Simple W-Potential

algorithm terminates when all $\mathcal{GW}_{empty}$ has been explored. A more formal description of the algorithm is given in Figure 4. At every point $\mathbf{x} \in \mathcal{GW}_{empty}$, the resulting W-potential is equal to the L^1 length of the minimal length path connecting $\mathbf{x}$ to $\mathbf{x}_{goal}$ through $\mathcal{GW}_{empty}$. It has no other local minimum than $\mathbf{x}_{goal}$.

This algorithm is similar in structure to the algorithm computing d_1. Its time complexity is also linear in the number of points of $\mathcal{GW}$ and constant in the number and shape of the obstacles. Figure 5 displays contours of $\mathbf{U}_p$ for the 2D workspace of Figure 1, with $\mathbf{x}_{goal}$ located in the upper lefthand corner of the bitmap array. The computation was done in a fraction of a second.

A property of the $\mathbf{U}_p$ function computed as above is the following. By tracking the flow of the negated gradient vector field $-\vec{\nabla}\mathbf{U}_p$ from any initial point $\mathbf{x}_{init}$, we obtain a path that connects $\mathbf{x}_{init}$ to $\mathbf{x}_{goal}$. In addition, this path is the shortest path for the L^1 distance (at the resolution of the bitmap array). In a 3D workspace, this computation may be preferable to exact methods, since the problem of computing the exact shortest distance in a polyhedral space is known to be NP-hard in the number of vertices under any L^p metric [Canny, 1987].

Notice that the algorithm given above actually computes $\mathbf{U}_p$ only in the connected subset of $\mathcal{GW}_{empty}$ that contains the goal position $\mathbf{x}_{goal}$. Hence, if the initial position $\mathbf{x}_{init}$ of p is such that $\mathbf{U}_p(\mathbf{x}_{init}) = M$, we can immediately return that there is no path connecting $\mathbf{x}_{init}$ to $\mathbf{x}_{goal}$. Step 2 of the above algorithm can easily be modified to accommodate the case where the goal of p is a subset of $\mathcal{W}$.

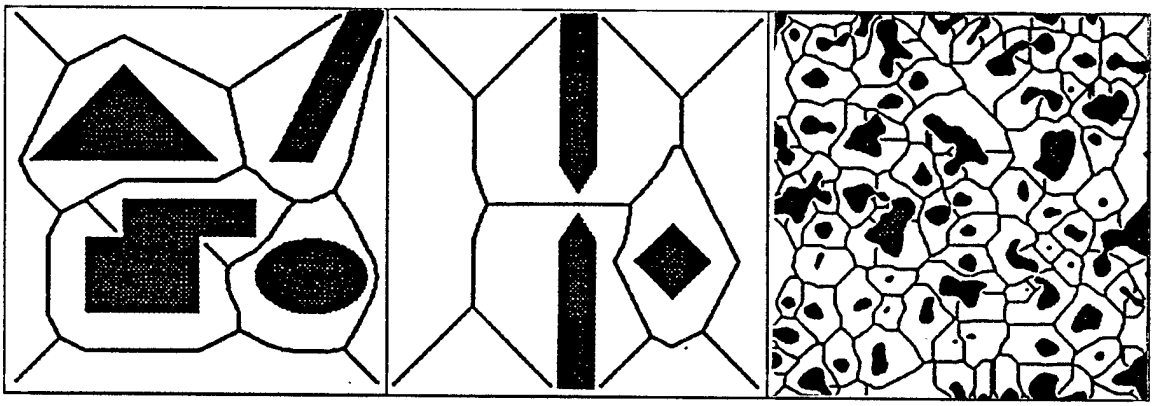


Figure 6: Examples of Workspace Skeletons

3.2.2 Improved W-Potential

A significant drawback of the simple W-potential is that it induces paths which typically graze obstacles in the workspace. Since several W-potentials usually have to be combined into a C-potential function $\mathbf{U}$ attracting the robot toward a goal configuration, the individual W-potentials may compete in such a way that they produce local minima of $\mathbf{U}$. In order to reduce the risk of creating local minima and to enlarge the maneuvering space of the robot, so that, if local minima are created, they can be more easily escaped, we propose an improved W-potential. This potential, like the previous one is free of local minima. In addition, its negated gradient pulls the control point p toward its goal position along a path of non-minimal length, which stays as far away as possible from the obstacles. This improved W-potential is computed in three stages.

First, the discrete L^1 distance $d_1(\mathbf{x})$ from every point $\mathbf{x} \in \mathcal{GW}_{empty}$ to the obstacles is computed and a $(d-1)$D subset $\mathcal{R}$ of $\mathcal{GW}_{empty}$ from the map d_1 is concurrently extracted. This subset is called the **workspace skeleton**. Second, the function $\mathbf{U}_p$ in the skeleton $\mathcal{R}$ is computed. Third, $\mathbf{U}_p$ is computed in the rest of $\mathcal{GW}_{empty}$.

The distance $d_1(\mathbf{x})$ is computed using the algorithm given in Figure 2. The workspace skeleton $\mathcal{R}$ is extracted during the computation of d_1 as the set of points where the "waves" – the lists L_i in the algorithm – issued from the boundary points of $\mathcal{GW}_{empty}$ meet. This is done by propagating not only the values of d_1, but also the points in the discretized boundary of $\mathcal{GW}_{empty}$ that are at the origin of the propagation. For each point $\mathbf{x}$ in $\mathcal{GW}_{empty}$, let $O(\mathbf{x})$ denote this point. Notice that $d_1(\mathbf{x})$ is the L^1 distance from $\mathbf{x}$ to $O(\mathbf{x})$. At step 2, we set $O(\mathbf{x})$ to $\mathbf{x}$ for every $\mathbf{x}$ in L_0. At step 3, let $\mathbf{x}$ be the point currently considered in L_i and $\mathbf{y}$ one of its neighbors. If $d_1(\mathbf{y}) = M$, we simply set $O(\mathbf{y})$ to $O(\mathbf{x})$; otherwise, if the L^1 distance between $O(\mathbf{y})$ and $O(\mathbf{x})$ is greater than some threshold (typically, 2), we include $\mathbf{y}$ in $\mathcal{R}$.

Figure 6 displays the skeletons computed for several 2D workspaces, including the workspace shown in Figure 1. Each of these skeletons is a kind of generalized Voronoi diagram of $\mathcal{W}_{empty}$ [Lee and Drysdale, 1981] for the L^1 distance. It is also similar to the skeleton ex-

1. (*Initialization.*)

   For every $\mathbf{x} \in \mathcal{GW}_{empty}$, set $\mathbf{U}_p(\mathbf{x})$ to infinity (i.e., a large number M). Set $\mathcal{S}$ to $\mathcal{R}$.

2. (*Computation of the augmented skeleton $\mathcal{S}$ by connecting the goal to the skeleton.*)

   Set $\mathbf{x}$ to $\mathbf{x}_{goal}$. While $\mathbf{x} \notin \mathcal{R}$, select a neighbor $\mathbf{y}$ of $\mathbf{x}$ having the largest value of d_1, include $\mathbf{y}$ in $\mathcal{S}$ and set $\mathbf{x}$ to $\mathbf{y}$.

3. (*Computation of the W-potential in the augmented skeleton.*)

   Set $\mathbf{U}_p(\mathbf{x}_{goal})$ to 0, L to $(\mathbf{x}_{goal})$ and $\mathcal{S}'$ to the empty list. (L is a list of points sorted by decreasing values of d_1.) Until L is empty, do: remove the first element $\mathbf{x}$ of L and insert it at the tail of $\mathcal{S}'$; for every d-neighbor $\mathbf{y}$ of $\mathbf{x}$ in $\mathcal{S}$, if $\mathbf{U}_p(\mathbf{y}) = M$ then set $\mathbf{U}_p(\mathbf{y})$ to $\mathbf{U}_p(\mathbf{x}) + 1$ and insert $\mathbf{y}$ in L. (At the end of this step, $\mathcal{S}'$ contains all the points in $\mathcal{S}$ sorted by increasing values of $\mathbf{U}_p$.)

4. (*Computation of the W-potential in the rest of $\mathcal{GW}_{empty}$.*)

   Set L_0 to $\mathcal{S}'$. For $i = 0, 1, 2, ...$, until L_i is empty, do: initialize L_{i+1} to the empty list; for every point $\mathbf{x}$ in L_i, for every neighbor $\mathbf{y}$ of $\mathbf{x}$ in $\mathcal{GW}_{empty}$, if $\mathbf{U}_p(\mathbf{y}) = M$ then set $\mathbf{U}_p(\mathbf{y})$ to $\mathbf{U}_p(\mathbf{x}) + 1$ and insert $\mathbf{y}$ at the end of L_{i+1}.

Figure 7: Algorithm Computing the Improved W-Potential

tracted from a region in a digitized image using techniques from Mathematical Morphology [Serra, 1982].

The improved W-potential in $\mathcal{R}$ is computed as follows (see Steps 2 and 3 in Figure 7). First, the goal position $\mathbf{x}_{goal}$ is connected to $\mathcal{R}$ by a path M following the gradient of d_1. The set $\mathcal{S} = \mathcal{R} \cup M$, called the "augmented skeleton", is connected. Then, all the points in $\mathcal{S}$ are labelled, starting at $\mathbf{x}_{goal}$. The label 0 is assigned to $\mathbf{x}_{goal}$; the label 1 is assigned to every d-neighbor[6] of $\mathbf{x}_{goal}$ in $\mathcal{S}$ which is at maximum distance d_1 of the obstacles; the label 2 is assigned to every d-neighbor of a previously labelled point that is at maximum distance d_1 of the obstacles; etc ... The algorithm terminates when all the points in the augmented skeleton have been labelled. At each iteration, the set of labelled points of $\mathcal{S}$, whose neighbors have not been analyzed yet, is sorted by decreasing values of d_1 in a height-balanced tree [Aho, Hopcroft and Ullman, 1983], so that both the insertion of a new point and the removal of a point of maximum value of d_1 are done in logarithmic time.

The W-potential at all the other points in $\mathcal{GW}_{empty}$ is computed using a wavefront expansion algorithm (see Step 4 in Figure 7) starting from $\mathcal{S}$, similar to the algorithms computing the d_1 map and the simple W-potential. The W-potential at each 1-neighbor $\mathbf{y}$ of every point $\mathbf{x}$ in $\mathcal{S}$ is set to $\mathbf{U}_p(\mathbf{x}) + 1$. The W-potential at the neighbors of these neighbors is iteratively incremented until all the points in $\mathcal{GW}_{empty}$ have been explored.

Figure 8 shows the equipotential contours of the resulting W-potential for the workspace

[6] In order to track the augmented skeleton reliably, it is more appropriate to use the d-neighborhood.

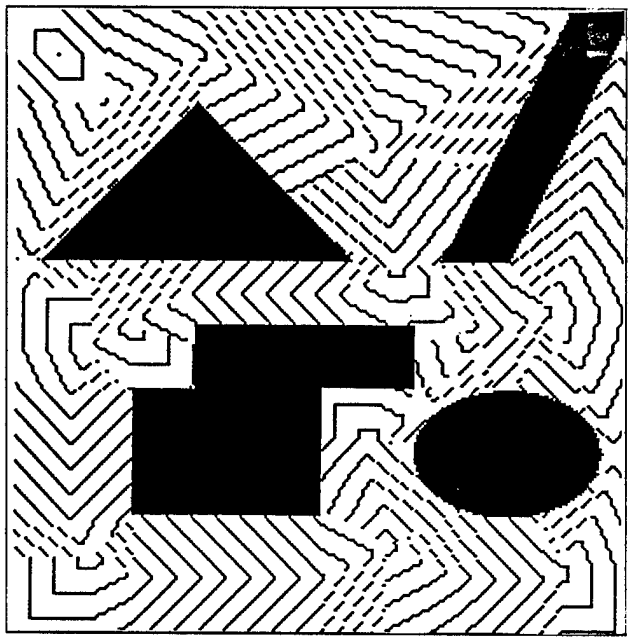


Figure 8: Contours of Improved W-Potential

of Figure 1. This W-potential generates no stable equilibrium state. Following its negated gradient from any initial position $\mathbf{x}_{init}$ produces a path that first connects $\mathbf{x}_{init}$ to $\mathcal{R}$, then stays as far as possible from the obstacles by following the safest curve of $\mathcal{R}$, and finally connects $\mathcal{R}$ to $\mathbf{x}_{goal}$. The above algorithm only computes $\mathbf{U}_p$ over the connected subset of $\mathcal{GW}_{empty}$ that contains $\mathbf{x}_{goal}$. It can be adapted to the case where the goal of p is a region in $\mathcal{W}$.

The complexity of computing the improved W-potential is slightly higher than for the simple one. Let a be the number of points in the bitmap array, and b the number of points in the augmented skeleton $\mathcal{S}$. The complexity of the algorithm is $O(a + b \log b)$. For reasonable workspaces, however, we have $b \propto a^{\frac{d-1}{d}}$ since the skeleton is a (d-1)D subset of the workspace. For such workspaces, the complexity is $O(a + a^{\frac{d-1}{d}} \log a)$, hence is linear in a. For the 256^2 workspace of Figure 8, the computation took about 2 seconds, including the computation of d_1 and $\mathcal{R}$.

Variants of the above W-potential are easy to define and compute. For example, in step 4 of the algorithm, we could compute $\mathbf{U}_p(\mathbf{y}) = \mathbf{U}_p(\mathbf{x}) + 1/d_1(\mathbf{x})$ and obtain a W-potential that becomes infinite on the boundary of $\mathcal{GW}_{empty}$. We will not detail the cosmetics of the computation of numerical W-potentials further. Our point is simply to show that a large family of W-potentials with various properties can be built within the bitmap numerical framework.

3.2.3 Remarks

The computation of d_1 (if the simple W-potential is used) and d_1 and $\mathcal{R}$ (if the improved W-potential is used) is not local and therefore must be done prior to the execution of the rest of any planning algorithm based on our path planning approach. However, the time complexity of the algorithm is only a function of the number of points in $\mathcal{GW}$. It is constant in both the number and the shape of obstacles, and the number of DOF's of the robot. Its implementation turns out to be quite fast.

From a conceptual point of view, neither the choice of the L^1 metric, nor the precise definition of the skeleton $\mathcal{R}$ is fundamental to the rest of our path planning approach. We could have used the more classical L^2 distance instead and computed the generalized Voronoi diagram for that metric. However, as we just noticed, the construction of the workspace potentials over $\mathcal{GW}_{empty}$ is a necessary precomputation before the rest of our path planning approach can be executed. The computation of the L^1 distance is faster than the computation of the L^2 distance. Notice also that, unlike the above computation of $\mathcal{R}$, the time complexity of constructing the algebraic description of the L^2 Voronoi diagram of a polygonal workspace increases with the number of vertices of the obstacles [Leven and Sharir, 1987]. We experimented with variants of the improved W-potential using slightly different constructions of the workspace skeleton $\mathcal{R}$. In general, these variants produced similarly satisfactory results.

3.3 Computation of C-Potentials

The C-potential $\mathbf{U}$ is computed as a combination:

$$\mathbf{U}(\mathbf{q}) = G(\mathbf{U}_{p_1}(X(p_1, \mathbf{q})), \ldots, \mathbf{U}_{p_s}(X(p_s, \mathbf{q})))$$

of the W-potentials $\mathbf{U}_{p_i}$, $i = 1, ..., s$, defined for several control points p_i. This combination concurrently attracts the different points p_i towards their respective goal positions. $\mathbf{U}(\mathbf{q})$ attains its minimal value, i.e. 0, when all the control points are at their goal positions. At any other configuration, it is strictly positive.

The control points are those used to input the description of the goal configuration of the robot. (By definition, the robot is at a goal configuration whenever all the control points are at their goal locations.) The goal configuration may not be uniquely defined. For instance, if $\mathcal{A}$ is a 2D object that can both translate and rotate in the plane (3D configuration space), the specification of the goal positions of two control points uniquely determines the goal configuration of the robot. If $\mathcal{A}$ is, say, a 10-DOF manipulator arm, specifying the desired positions of some points in the end-effector determines a goal region in configuration space. In all cases, we denote by $\mathcal{C}_{goal}$ the subset of goal configurations.

It is important that a path planner allows to specify a goal configuration by specifying the goal positions of a small number of points in the robot. Indeed, in many tasks, the goal configuration is incompletely determined by the task constraints. Arbitrarily selecting one configuration among the various possible ones may result in a more difficult, even impossible, path planning problem. Furthermore, if the robot has many DOF's, specifying a unique

goal configuration is a difficult task in itself, since it requires a collision-free placement of the various bodies of the robot to be found.

In order to precisely define the C-potential, we must now specify the arbitration function G. In most of the previous systems using the artificial potential field approach, G has been chosen as a positive linear combination of the W-potentials [Khatib, 1986]:

$$G(y_1, \ldots, y_k) = \sum_{i=1}^{i=k} \lambda_i y_i.$$

This simple choice seems natural because it does not favor one control point over the others. However, precisely for that reason, conflicts among the points tend to be frequent, producing numerous spurious local minima.

The choice of the function G is specially important because it highly influences the number and the depths of the local minima of $\mathbf{U}$. With W-potentials free of local minima, the workspace concavities do not directly create local minima of $\mathbf{U}$. It is the concurrent attraction of the different control points toward their respective goal positions, which creates these local minima. This results from the fact that these points do not move independently from each other. As discussed above, the function G determines how the competition between the different points is to be regulated.

The choice of G which seems to minimize the number of local minima is:

$$G(y_1, \ldots, y_k) = \min_{i=1}^{i=k} y_i.$$

Indeed, this arbitration function favors the attraction of the point which is already in the best position to reach its goal. However, when one point has reached its goal position, the potential field is identically zero, and the other points are not attracted toward their goal positions. A way to avoid this shortcoming is to add another term to the arbitration function:

$$G(y_1, \ldots, y_k) = \min_{i=1}^{i=k} y_i + \epsilon \max_{i=1}^{i=k} y_i \tag{1}$$

where ϵ is a small real number. In our experiments, we used $\epsilon = 0.1$. However, the best value of ϵ may depend on the robot.

Another choice for G is:

$$G(y_1, \ldots, y_k) = \max_{i=1}^{i=k} y_i. \tag{2}$$

This arbitration function favors the attraction of the control point that is the furthest away from its goal (along the path determined by the W-potential). It tends to increase the number of competitions among the control points and, therefore, the number of local minima. However, it can be a good choice for robots with many DOF's. As a matter of fact, the number of local minima is not the only measure for the quality of the C-potential. Another critical factor is the depths of these minima. Indeed, in some cases, it may be preferable to have several small local minima, rather than a single, very deep one. This is specially true when the robot has many DOF's, since the number of discretized configurations contained in a local minimum well of given depth increases exponentially with

the dimension of the configuration space. Various experiments with the above arbitration function indicate that in general it increases the number of local minima, but reduces their volumes.

Obviously, many other arbitration functions can be imagined. In some of our experiments, we used other C-potentials, as indicated further in the paper.

We now describe four path planning techniques incorporating the general planning approach presented in the introduction. All these techniques iteratively consider each level of resolution in the workspace and configuration space pyramids, from the coarsest to the finest. They terminate with success as soon as a path has been generated. They return failure if, after having considered the finest bitmaps, they still have not generated a path. Below, we only describe the algorithms executed at each level of resolution. We denote by $\mathcal{GW}$ and $\mathcal{GC}$ the current workspace and configuration space grids.

4 Best-First Motion Technique

4.1 Description

We start with a very simple path planning algorithm[7]. This algorithm essentially performs a best-first search [Nilsson, 1980] of the corresponding configuration space discretized grid $\mathcal{GC}$ using the C-potential $\mathbf{U}$ as the cost function to minimize. This means that the possible successors of a configuration are all its k-neighbors, for some $k \in [1, n]$, in the grid and these neighbors are selected by decreasing order of the C-potential. A configuration has $2n$ 1-neighbors, $2n^2$ 2-neighborhoods,..., and $3^n - 1$ n-neighbors. In our implementation of the best-first motion technique, we used $k = n$. The size of the neighborhood thus increases exponentially with n, but as we will see below, the planing technique is intrinsically limited to planning problems involving few DOF's (typically, $n \leq 4$), so that the size of the maximal neighborhood remains reasonable.

As long as the algorithm does not reach a local minimum of the C-potential, the search reduces to following an approximation of the negated gradient of the C-potential (fastest descent procedure). When a local minimum is reached, the same algorithm naturally fills up the local minimum well until a saddle point is reached. Then, it proceeds again along the negated gradient of $\mathbf{U}$. It stops when a goal configuration ($\mathbf{U} = 0$) is attained. Hence, the algorithm basically searches a graph connecting local minima among them, but this graph is not explictly represented. Between two minima, it simply executes a deterministic gradient motion. At each local minimum it spends so time filling it up. Two local minima are adjacent in the local minima graph if they communicates through a saddle point. From a local minimum, the algorithm always goes to the adjacent local minimum attainable through the lowest saddle point. "Backtracking" occurs in the local minima graph when two adjacent minima get filled up; then, they are implicitly merged into a single minimum, their common filling level being the new height of the minimum.

The algorithm is resolution-complete, i.e. it is guaranteed to reach the goal in a finite amount of time whenever a solution path exists (at the maximal resolution), or return

[7]This algorithm is described in more detail in [Barraquand and Latombe, 1989].

failure when there is no solution. In most cases (virtually all reasonable cases), a very small subset of the grid $\mathcal{GC}$ has to be explored before the algorithm terminates. It is easy, however, to create a problem admitting no solution path, which requires most of $\mathcal{GC}_{free}$ to be explored before the planner gives up.

At this stage, there is an important aspect of the algorithm that must be made precise. Since the C-potential only depends on the distance of a few control points to the obstacles, the best-first search does not guarantee that collisions of the robot with the obstacles will be completely avoided. Therefore, whenever the planner considers a new configuration in $\mathcal{GC}$, it should check that it lies in the free space. Because the planner does not represent the C-obstacles explicitly, the verification is done in the workspace as explained below.

As the workspace representation is distributed, it seems natural to represent the robot $\mathcal{A}$ in the same way, i.e. by a bitmap. If we were using a massively parallel computer, this representation would be the simplest and the most efficient to test collisions. For example, we could have one processor per point in the bitmap representation of $\mathcal{A}$; this processor would compute the position of the point in the workspace at the current configuration of $\mathcal{A}$ and determine whether it is contained in the obstacle region, or not. However, as we implemented our planner on a sequential computer, we chose to represent the boundary of $\mathcal{A}$ algebraically.. The implemented collision checking technique is a classical "divide-and-conquer" algorithm, which consists of iteratively adjusting the number of points in the robot's boundary which are necessary to check collisions reliably. To illustrate this idea, let us consider a simple robot modelled as a single straight line segment of length L. The function d_1 has already been computed over $\mathcal{W}_{empty}$. Instead of checking the distance d_1 for each discretized point on the line segment modelling $\mathcal{A}$ and then calculating the minimum of all these distances, we first compute the distances $d_1(begin(\mathbf{q}))$ and $d_1(end(\mathbf{q}))$ of the two extremity points *begin* and *end* of the segment. If $\min\{d_1(begin(\mathbf{q})), d_1(end(\mathbf{q}))\} > L$, then we are certain that the robot does not collide any obstacle. Otherwise, we cut the segment in two half-length segments, and we recursively apply the procedure to the two new segments. If the robot boundary is modelled by a polygon, the same procedure can be applied to each edge of the polygon. The procedure can also be generalized to higher-order shapes of the robot boundary such as circular and elliptical arcs. When the robot is far from the obstacles, very few points on its boundary need to be checked. When the robot gets closer to an obstacle, an increasing number of points (up to the current resolution of the workspace bitmap) in the boundary segment that is close to make contact have to be checked.

The search algorithm only considers the neighbors of a configuration that are collision-free. Hence, there may be two types of local minima: the natural minima of $\mathbf{U}$ (where the gradient is zero) and the minima located at the boundary of $\mathcal{GC}_{free}$ (where the gradient is not zero in general). When an obstacle is hit, the last configuration before the collision (i.e., in the boundary of $\mathcal{GC}_{free}$) is taken as the local minimum. Both types of minimum are escaped in the same fashion.

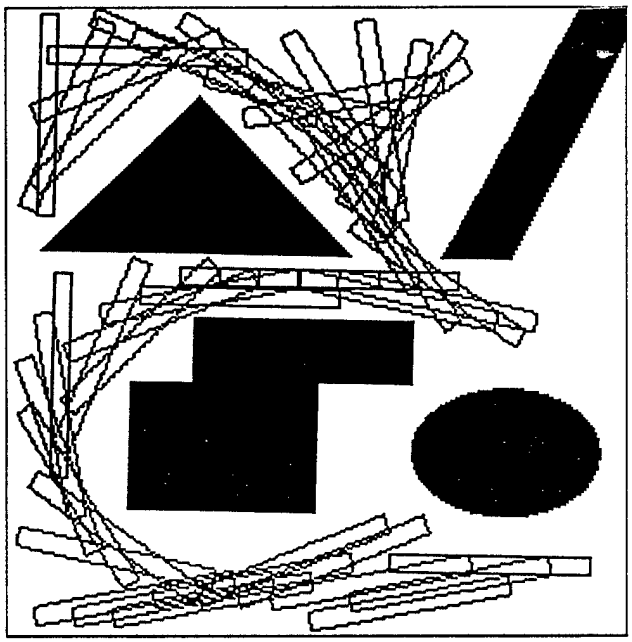


Figure 9: Path Generated for a 3-DOF Mobile Robot

4.2 Experimental Results

We experimented the path planner on a "mobile robot" with two DOF's of translation and one DOF of rotation, namely a long rectangular bar in a 2D workspace. Figure 9 shows an example of path generated by the planner. This path demonstrates the ability of the planning technique to produce complex maneuvers. The running time to produce the path was 1 second[8]. This is three orders of magnitude faster than the running times reported in [Brooks and Lozano-Pérez, 1983] for similar (though apparently simpler) path planning problems. One order of magnitude is due to the faster computer we used. The other two are actually a product of our algorithm.

Figure 10 shows another example of the planning abilities of the algorithm. For the same robot, a path was generated in less than 5 seconds within a 512^2 bitmap representing a workspace cluttered by more than 70 complex-shaped, randomly generated obstacles. This example would probably be very difficult to run with a planner using a semi-algebraic representation of the workspace, e.g. an exact cell decomposition planning technique. It demonstrates the power of the "distributed" representations used in our planner relatively to the "centralized" algebraic representations.

In both examples, the C-potential was computed by considering two control points located at the midpoints of the two small edges at the extremities of the bar, using the improved W-potential described in Subsection 3.2.2. The C-potential was computed using the arbitration function defined by formula (1). At first sight, the experimental efficiency of the best-first

[8]Since there may be many simple variants of the techniques presented in this paper, only the order of magnitude of the given execution times is actually pertinent.

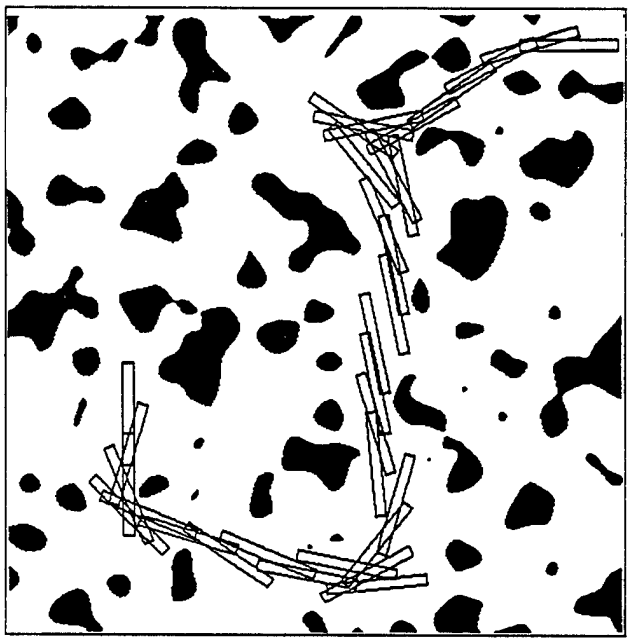


Figure 10: Path Generated Among Randomly Distributed Obstacles

motion technique is surprising. The fundamental reason of this efficiency is the following. The potential fields computed in the workspace are designed to be "perfect" potential fields for a point robot, in the sense that they have no other minimum than the goal. Therefore, complex shaped obstacles with large concavities do not directly create local minima of the C-potential. Local minima occur only where the workspace is so cluttered that the solution path has to come very close to the obstacles. But in that case, the local minima are close to the boundary of the free space. Hence, they have small domains of attraction, i.e. contain a small number of discrete configurations. The algorithm fills the wells very quickly. Furthermore, because it guides the robot along a path where maneuvering space is maximized, the improved W-potential tends to produce fewer local minima. The algorithm also works in a satisfactory fashion, but is in general slower, when the simple W-potential is used in place of the improved one.

However, in practice, the above planning technique is only applicable to robots with a small number n of DOF's – typically, $n \leq 4$. Indeed, the number of discrete configurations in a local minimum well increases exponentially with the number of DOF's. Filling up a well would be too time consuming for robots with many DOF's.

5 Random Motion Technique

5.1 Description

The path planning algorithm[9] presented in this section essentially differs from the previous one in the way it escapes local minima. Rather than filling up each encountered local min-

[9]This algorithm is described in more detail in [Barraquand and Latombe, 1989].

imum, it applies a Monte-Carlo procedure, which consists of generating Brownian motions until the minimum is escaped.

Starting from the initial configuration, the algorithm first searches the current grid $\mathcal{GC}$ in a best-first fashion. Thus, it follows the negated gradient of the C-potential, until it reaches a local minimum (call it $\mathbf{q}_{loc}$). We call such a motion a **gradient motion**. If $\mathbf{q}_{loc} \in \mathcal{C}_{goal}$, the planner returns the solution path generated. Otherwise, it attempts to escape the local minimum by generating several **random motions** from $\mathbf{q}_{loc}$. These random motions are Brownian motions described in more detail in the next subsection.

The generation of gradient motions and the recognition of local minima are done using the n-neighborhood in $\mathcal{GC}$. As long as $n \leq 4$, this raises no difficulty. However, when n becomes too big (and we ran the random motion technique for robots with many DOF's), the size of the n-neighborhood becomes too large. In this case, at each step of the gradient motion, only a few points in the n-neighborhood of the current position $\mathbf{x}$ are considered. This is done by exploring the n-neighborhood of $\mathbf{x}$ iteratively in a random order and limiting the number of iterations. At each iteration, a point $\mathbf{x}'$ is randomly chosen in the n-neighborhood of $\mathbf{x}$ (using a uniform distribution law). If $\mathbf{U}(\mathbf{x}') < \mathbf{U}(\mathbf{x})$, $\mathbf{x}'$ is taken as the successor of $\mathbf{x}$ along the path of the gradient motion (hence, the motion may only follow a rough approximation of the negated gradient flow). The number of iterations is limited to a few tens to a few hundreds (depending on the value of n). If none of them generates a successor of $\mathbf{x}$, $\mathbf{x}$ is treated as a local minimum. An alternative to this technique would be to use a smaller neighborhood, for instance the linear 1-neighborhood. In fact, we tried several alternatives, and the technique described above gave the best results. In particular, due to the crude discretization that it entails, a small neighborhood often resulted in the detection of many fictitious local minima.

At the terminal configuration of every random motion, the algorithm executes a gradient motion until it reaches an hopefully new local minimum. From each local minimum, if none of them is in the goal region, it performs another set of random motions. Etc... The graph of the local minima is thus incrementally built, the path joining two "adjacent" local minima being the concatenation of a random motion and a gradient motion. Using the values of the C-potential at the local minima, a best-first search of this graph is performed until the goal configuration is reached or all the local minima in the graph have been expanded (i.e., a series of random motions have been executed from them). When a solution path is generated, it is smoothed using a classical variational calculus technique. An interesting property of this algorithm is that all the random motions starting from a given local minimum can be performed concurrently on a parallel machine, since there is no need for communication between the different processing units.

Many variants of the best-first search scheme are possible, and we experimented with several ones. The search technique that gave the best results consists of iteratively executing random motions starting at the current local minimum $\mathbf{q}_{loc}$ and, after each one, performing a gradient motion. If the attained local minimum has a lower C-potential than $\mathbf{q}_{loc}$, the iteration is stopped and the search proceeds from this new local minimum, by executing new random motions. The number of random motions starting at each local minimum is arbitrarily bounded. If no random motion from $\mathbf{q}_{loc}$ followed by a gradient motion leads

the planner to a better minimum, $\mathbf{q}_{loc}$ is considered as a dead-end. The algorithm then backtracks to a point in the current path connecting the initial configuration to $\mathbf{q}_{loc}$. This point is selected randomly, using a uniform distribution law over the set of points contained in the current path. The search for a path resumes at that point by executing a gradient motion. Since the backtracking point may belong to the subpath generated by a random motion, the gradient motion may lead to a local minimum not encountered yet. This search technique has the advantage of not requiring an explicit representation of the local minima graph to be maintained.

As the algorithm uses a random procedure to build the graph of the local minima, it is not guaranteed to find a path whenever one exists. In other words, the algorithm is not complete. However the properties of Brownian motions make it possible to prove that when the number of Brownian motions executed from every local minimum is unbounded (the computation time may tend towards infinity), the probability to reach the goal converges towards 1 (see [Barraquand and Latombe, 1989]). Hence, we say that the algorithm is **probabilistically resolution-complete**. However, this convergence-in-distribution property, which is well-known for the so-called "simulated annealing" algorithms[10] (see [Geman and Geman, 1986]), is a very weak one. Indeed, the totally uninformed algorithm, which executes a Brownian motion from $\mathbf{q}_{init}$ and terminates when it enters a small neighborhood of the goal configuration, is also probabilistically resolution-complete!

A weakness of the algorithm with the search technique described above is that it may be unable to recognize that a planning problem admits no solution path. However, since the algorithm turns out to be relatively fast in most practical cases where there exist solution paths (see Subsection 5.3), one way to deal with this drawback is to arbitrarily limit the computing time allocated to the planner. By simplifying a bit, we can say that we traded decidability and slow computation for semi-decidability and fast computation.

5.2 Brownian Motions

In theory, a Brownian motion is a continuous stochastic process. In our planner, it is implemented as a discrete random walk. It consists of discretizing time into units and executing a motion step, whose projection along each axis q_i, $i = 1, ..., n$, is randomly Δ_i or $-\Delta_i$, at each time unit. The motion lasts a certain number of time units, say t. At each time unit, the motion step is independent of the previously executed steps. This random walk is known to converge almost surely toward a Brownian motion when every Δ_i tends toward 0 [Papoulis, 1965].

Without lack of generality, let us take the local minimum, $\mathbf{q}_{loc}$, as the origin. The configuration attained by a Brownian motion of duration t is a random variable $\mathbf{Q}(t) = (Q_1(t), ..., Q_n(t))$, with the following properties [Papoulis, 1965]:

- The density $p_i(q_i)$ of $Q_i(t)$ is:

$$p_i(q_i) = \frac{1}{\Delta_i\sqrt{2\pi t}} \exp(-\frac{q_i^2}{2\Delta_i^2 t}).$$

[10] For a comparison between simulated annealing algorithms and our planning algorithm, see [Barraquand and Latombe, 1989].

- The standard deviation D_i of $Q_i(t)$ is:

$$D_i = \Delta_i \sqrt{t}.$$

The Brownian motion is well-defined as long as it does not encounter any obstacle in configuration space. When it hits the boundary of a C-obstacle, the Brownian motion has to be adapted so that it stays in the free space. The classical adaptation of a Brownian motion when the space is bounded consists of reflecting the motion that would take place if there were no boundary, symmetrically to the tangent hyperplane of the boundary at the collision configuration. The mathematical consistency of this adaptation is discussed in detail in [Anderson and Orey,1976]. Our planner, which does not construct an explicit representation of the C-obstacles, does not know the orientation of the tangent hyperplane at the collision configuration. Hence, whenever a motion step leads to a collision, instead of bouncing symmetrically on the hyperplane tangent to the C-obstacle, the planner guesses another random step and substitutes it for the previous one. The collision is detected in the workspace using the divide-and-conquer technique described in Subsection 4.1.

The duration t of a Brownian motion still has to be chosen. If it is too short, the motion has little chance to escape the local minimum. If it is too long, the planner may waste time and loose the opportunity of using the C-potential gradient information when it becomes useful again. We can define the **attraction radius** $A_{Ri}(\mathbf{q}_{loc})$ of any local minimum $\mathbf{q}_{loc}$ of $\mathbf{U}$ along each axis q_i as the distance along q_i between $\mathbf{q}_{loc}$ and the nearest saddle point of $\mathbf{U}$ in that direction. In order to escape the local minimum $\mathbf{q}_{loc}$, the minimum distance that the robot has to travel in each direction q_i from $\mathbf{q}_{loc}$ is precisely $A_{Ri}(\mathbf{q}_{loc})$. If we were able to estimate the statistics of A_{R_i}, the property $D_i = \Delta_i \sqrt{t}$ would give us a clue for escaping the minimum. The duration of the Brownian motion would be the function $t(\mathbf{q}_{loc})$ defined by:

$$t(\mathbf{q}_{loc}) \approx \max_{i \in [1,n]} \left(\frac{A_{Ri}(\mathbf{q}_{loc})}{\Delta_i} \right)^2. \tag{3}$$

But, as we make no assumption on the obstacle distribution, we cannot infer any strong statistical property about $\mathbf{U}$ and A_{R_i}. However, in general, we may consider that the distance A_{Ri} for each parameter q_i does not exceed the distance that would provoke a motion of the robot longer than the workspace diameter itself. Normalizing this diameter to 1, this assumption yields the following estimate of A_{Ri} for any local minimum $\mathbf{q}_{loc}$:

$$A_{Ri} \approx 1/J^i_{\text{sup}} \tag{4}$$

This estimate, combined with the fact that A_{R_i} is strictly positive, leads us to treat A_{R_i} as a random variable with a truncated Laplace distribution of density:

$$p(A_{Ri}) = J^i_{\text{sup}} \exp(-J^i_{\text{sup}} A_{Ri})$$

(The truncated Laplace distribution is the less informed distribution – i.e. the one which maximizes entropy – for a positive random variable of given expected value.)

We have $\Delta_i \approx \delta / J^i_{\text{sup}}$ (see Subsection 2.3), where δ is the distance between two adjacent points in $\mathcal{GW}$. Combining this formula with (3) and (4), we obtain:

$$t \approx \frac{1}{\delta^2}.$$

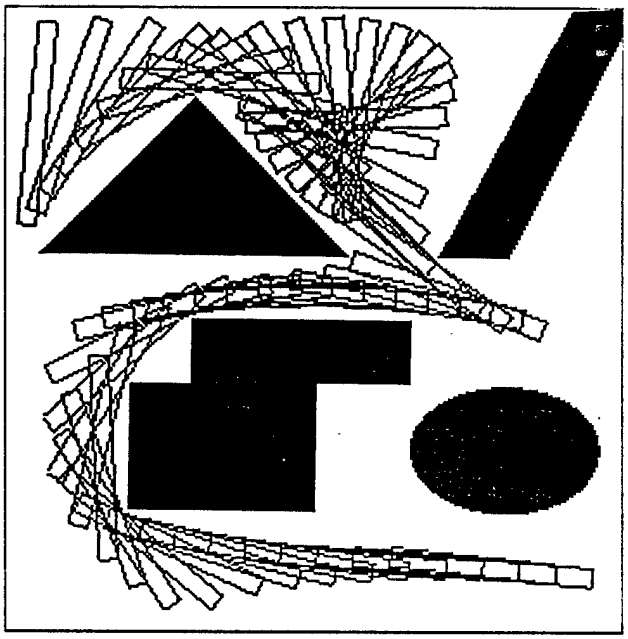


Figure 11: Path Generated for a 3-DOF Robot

One could take t equal to the above value. However, this choice would implicitly assume that all the attraction radii are the same, which is not the case. Using the above distribution of A_{R_i}, we choose t as the value of a random variable T with the following density:

$$p(t) = \frac{\delta}{2\sqrt{t}} \exp(-\delta\sqrt{t}).$$

One can verify that the expected value of T is indeed $1/\delta^2$.

In fact, a value of T gives a maximal duration of the Brownian motion. At each time unit during the motion, the planner checks the C-potential at the current configuration against $\mathbf{U}(\mathbf{q}_{loc})$. If it is smaller, the planner terminates the Brownian motion.

5.3 Experimental Results

We have tested the random motion technique with several different robot stuctures. We describe now some of the obtained results. Since the planning technique contains several random components, the running time for the same planning problem is not constant. The times given below are only typical times. It is not unusual that two running times for the same example differ in a ratio of 5. The execution times would be both smaller and much more stable on a parallel architecture allowing the concurrent execution of several random motions.

First, we applied it to the 3-DOF robot example shown in Figure 9 using the same C-potential as with the best-first motion algorithm. We got the path shown in Figure 11 (after smoothing). The overall computation time was 10 seconds, which is about ten times slower

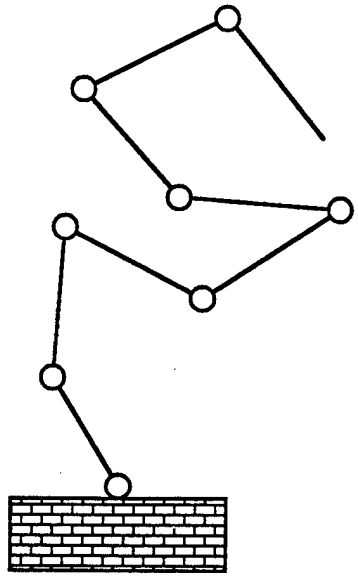


Figure 12: Structure of the 8-DOF Manipulator

than with the pure best-first search. However, we think that a parallel implementation of the random motion technique will considerably reduce this computation time.

We experimented with the planner on a 8-DOF serial manipulator with only revolute joints (see Figure 12). Figure 13 illustrates a path generated by the planner. The goal region is defined by the position of the end-point p of the last link of the robot. The C-potential $\mathbf{U}$ was computed as the improved W-potential at p, i.e. $\mathbf{U}_p(X(p, \mathbf{q}))$. The path was generated in 2 minutes, covering all the computation performed between the input of the robot and workspace models and the display of the generated path.

We also experimented with the planner using a 10-DOF non-serial manipulator arm with 3 prismatic and 7 revolute joints, as depicted in Figure 14. Figures 15 illustrates a path generated by the planner. The C-potential was computed by considering two control points located at the end-points of the two kinematic chains, using the improved W-potential field and the arbitration function defined in formula (2). The computation time was 3 minutes. (In this example, the number of configurations in the grid $\mathcal{GC}$ in which the path was found is of the order of $100^{10} = 10^{20}$.)

Finally, we have run the path planning algorithm with a 31-DOF serial manipulator carrying a bar in a 3D workspace. The manipulator (see Figure 16) consists of a sequence of 10 identical modules, each with three DOF's (one translation and two rotations). The last module provides a third rotation. A path generated by the planner is illustrated in Figure 17. The C-potential was computed by considering two control points located at the end-points of the bar, using the improved W-potential field and the arbitration function of formula (2). The computation time was 15 minutes. The level of resolution of the workspace representation required to find this path was 128^3.

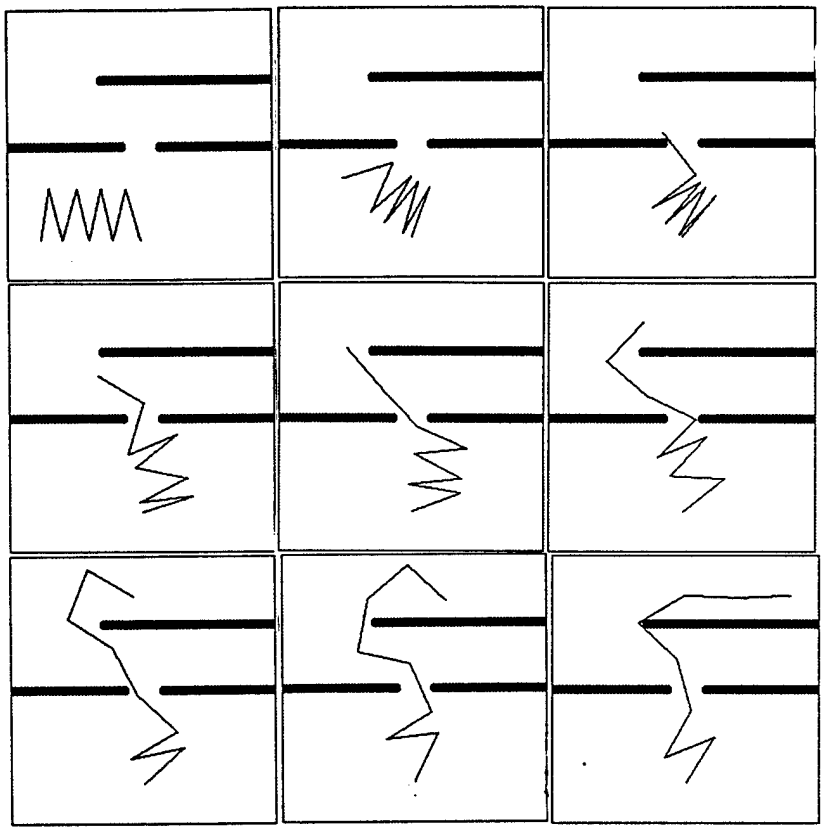


Figure 13: Path Generated for the 8-DOF Manipulator

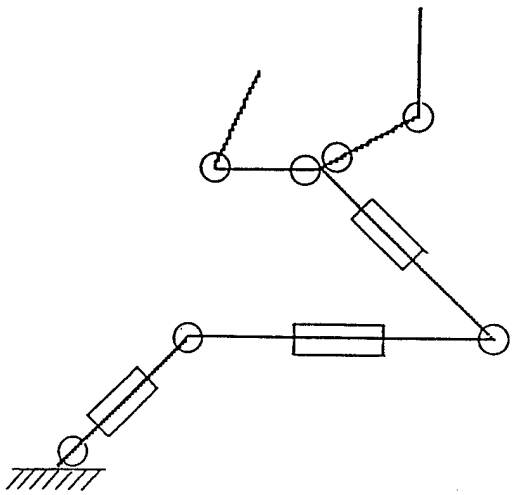


Figure 14: Structure of the 10-DOF Manipulator

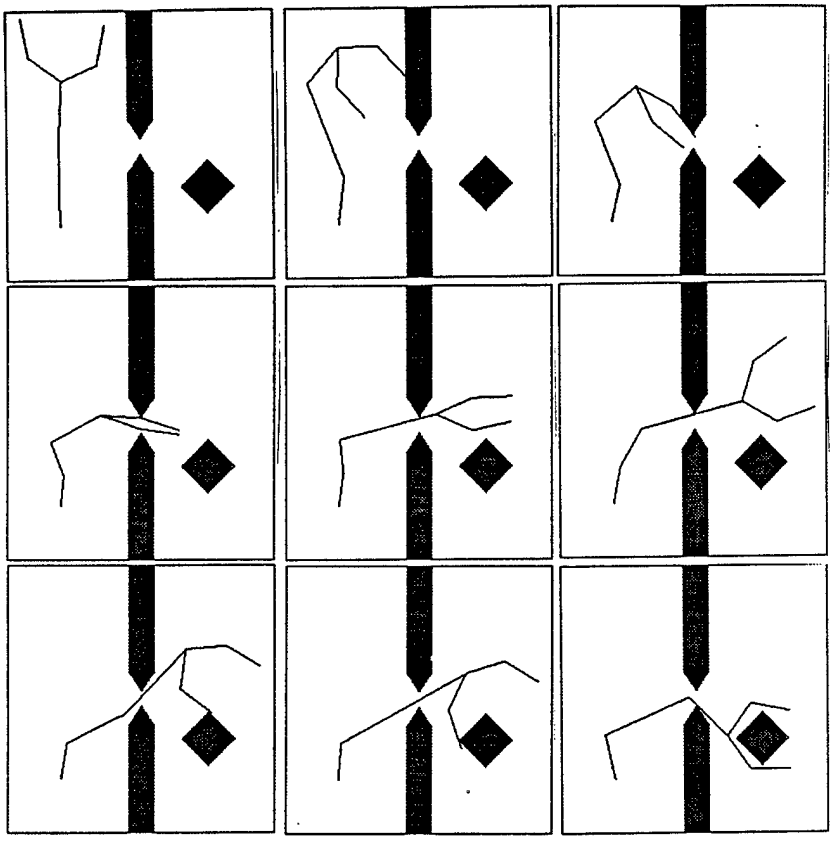


Figure 15: Path Generated for the 10-DOF Manipulator

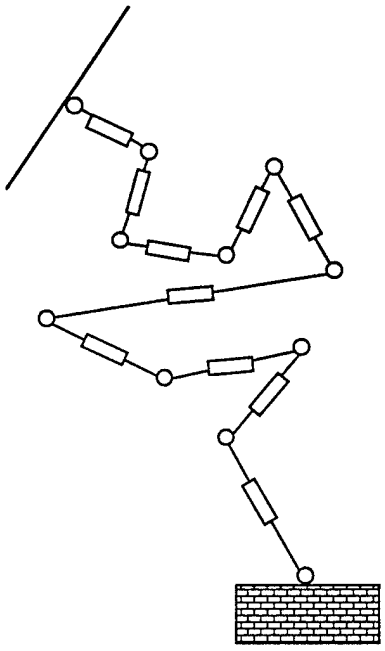


Figure 16: Structure of the 31-DOF Manipulator

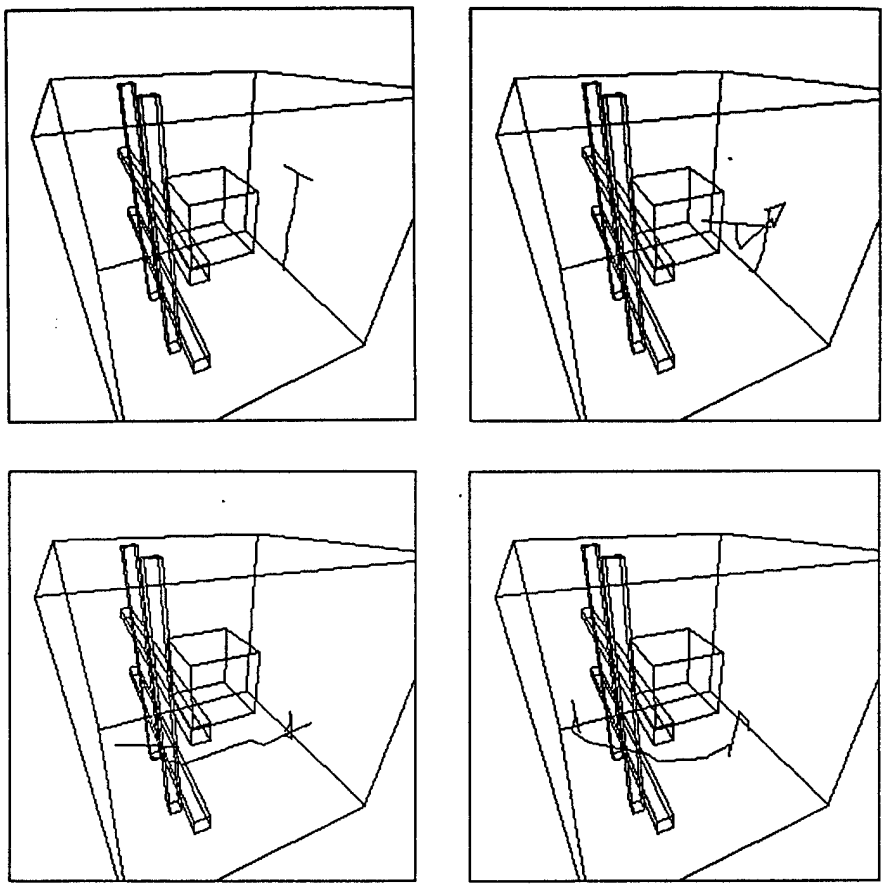


Figure 17: Path Generated for the 31-DOF Manipulator

In all the above multi-link robot examples, we simulated mechanical stops by limiting the range of displacement of every joint. Except with the 31-DOF arm, we also prohibited collisions among the links, by checking collisions between any two links using the technique of Subsection 4.1.

Again the efficiency of the algorithm may be surprising. We think that it results from the fact that a typical path planning problem has many solutions, so that a globally random process can find one, providing it is sufficiently informed most of the time. This efficiency result is not new. Monte-Carlo techniques have already been helpful for solving NP-complete problems. For example, Cerny [Cerny, 1985] described an algorithn that generates sub-optimal solutions to the traveling salesman problem for more than 10,000 towns. Kirkpatrick et al. [Kirkpatrick, Gelatt and Vecchi, 1983] proposed an algorithm for the placement and the routing of VLSI chips, which is better than human experts. In both problems, the very large search space is associated with a large number of "good" sub-optimal solutions.

6 Valley-Guided Motion Technique

6.1 Description

The path planning algorithm[11] described in this section differs significantly from the previous two. It consists of searching the "valleys" of a C-potential $\mathbf{U}$ in $\mathcal{C}_{free}$. The set of valley points of $\mathbf{U}$ is called the **valley roadmap** and is denoted by $\mathcal{V}$.

The algorithm is similar in structure to a retraction algorithm [Ó'Dúnlaing, Sharir and Yap, 1983]:

1. Given an initial and a goal configurations, $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, generate two new configurations, $\mathbf{q}_i$ and $\mathbf{q}_g$, local minima of $\mathbf{U}$, by following the negated gradient of $\mathbf{U}$ from $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, respectively.

2. Search the valley roadmap $\mathcal{V}$ for a path connecting $\mathbf{q}_i$ to $\mathbf{q}_g$.

3. If step 2 terminates successfully, return the path obtained by concatenating the three paths joining respectively $\mathbf{q}_{init}$ to $\mathbf{q}_i$ (gradient motion), $\mathbf{q}_i$ to $\mathbf{q}_g$ (valley-guided motion), and $\mathbf{q}_g$ to $\mathbf{q}_{goal}$ (gradient motion). Otherwise, return failure.

This algorithm does not require $\mathbf{U}$ to be minimum at the goal configuration. However, since $\mathbf{q}_{goal}$ is projected in $\mathcal{V}$ by a gradient motion, $\mathbf{q}_{goal}$ has to be uniquely defined, which was not the case with the previous two algorithms[12]. Ideally, $\mathbf{U}$ should be such that $\mathcal{V}$ is a 1D subset of the free space $\mathcal{C}_{free}$. In addition, $\mathbf{U}$ should "represent the topology" of $\mathcal{C}_{free}$, i.e. $\mathbf{q}_i$ and $\mathbf{q}_g$ should be connected in $\mathcal{V}$ iff $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ are connected in $\mathcal{C}_{free}$. In [Barraquand, Langlois and Latombe, 1989], we discuss the conditions under which a C-potential satisfies

[11] This algorithm is described in more detail in [Barraquand, Langlois and Latombe, 1989].

[12] Actually, one could extend the above algorithm to handle the case where a goal configuration is any configuration in a subset $\mathcal{C}_{goal}$ of the configuration space. But this would require each configuration in the discretized boundary of $\mathcal{C}_{goal}$ to be connected by a gradient motion to $\mathcal{V}$. Except for robots with few DOF's, this is likely to be impractical.

these two properties. Unfortunately, these conditions are quite involved and difficult to verify.

At step 2, $\mathcal{V}$ is searched in a depth-first manner. A decision is made at every crossroad using a simple heuristic function defined as another C-potential $\mathbf{U}_{heur}$. This C-potential is constructed as described in Section 3, i.e. by combining local-minima-free W-potentials. In our experiments, this heuristic potential dramatically reduced the computation times.

It now remains to specify how $\mathcal{V}$ is constructed. Unfortunately, as noticed in [Barraquand, Langlois and Latombe, 1989], there has not been much research on the concept of valley reported in the literature so far. One way to get a simple, but ad hoc, definition of valley points is to discretize the set of possible directions of the valleys. Namely, we force these directions to be the various q_i coordinate axes. Thus, we only have n possible valley directions. We define a configuration $\mathbf{q}$ to be a valley point along the q_i coordinate axis ($i \in [1, n]$) when all the values of the C-potential $\mathbf{U}$ at the $(2n-2)$ 1-neighbors in the hyperplane orthogonal to this direction are greater than $\mathbf{U}(\mathbf{q})$. According to this simple definition, every local minimum of $\mathbf{U}$ is also a valley point of $\mathbf{U}$ (which is consistent with our intuitive notion of valley), so that the valley roadmap $\mathcal{V}$ can be regarded as a graph connecting the local minima of $\mathbf{U}$. This graph has to be searched for a path between two minima, $\mathbf{q}_i$ and $\mathbf{q}_g$.

In order to check whether a point $\mathbf{q}$ is a valley point or not, we thus use the simple following algorithm:

1. Compute $\mathbf{U}(\mathbf{q})$.
2. Compute the $2n$ values of $\mathbf{U}$ at the 1-neighbors of $\mathbf{q}$.
3. For each possible valley direction $i \in [1, n]$ do:

   Compare $\mathbf{U}(\mathbf{q})$ to the $2n-2$ values of $\mathbf{U}$ at the 1-neighbors in the hyperplane orthogonal to the q_i axis. If $\mathbf{U}(\mathbf{q})$ is smaller or equal to these $2n-2$ values, $\mathbf{q}$ is a valley point.

The time complexity of this algorithm is $O(n^2)$.

As non-degenerated valleys are 1D, the safest neighborhood to track them is the n-neighborhood. Unfortunately, the cardinal of this neighborhood is $3^n - 1$, yielding an exponential time tracking algorithm. For instance, in a 10D configuration space, using the 10-neighborhood entails performing about 60,000 valley checkings at every increment of the tracking process. Instead, the implemented planner uses the 2-neighborhood, whose size is quadratic in n, allowing each tracking increment to be performed in $O(n^4)$ time. This compromise between time and completeness appeared reasonable in most of the experiments we made. Complex paths have been generated, meaning that valleys are trackable using the 2-neighborhood. Nevertheless, the choice of the 2-neighborhood is empirical.

Some valleys may be dead-ends. Valley tracking stops when the tracking algorithm is unable to proceed further. It also stops if the valley is running into an obstacle in configuration space (this can be detected by using the collision checking technique described in Subsection

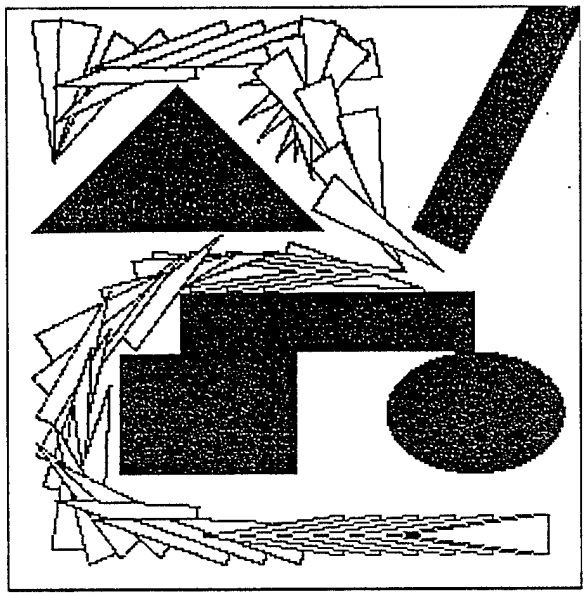


Figure 18: Path Generated for a 3-DOF Mobile Robot

4.1). In both cases, the planning algorithm backtracks to a crossroad point in the valley roadmap.

6.2 Experimentation

We experimented with the valley-guided motion technique on a variety of path planning problems[13]. The best results were obtained with the C-potential $\mathbf{U}$ defined as follows [Barraquand, Langlois and Latombe, 1989]:

$$\mathbf{U}(\mathbf{q}) = \sigma \log(\sum_{i=1}^{s} \exp(\mathbf{U}_{p_i}(X(p_i, \mathbf{q}))/\sigma))$$

where σ is a small number and the p_i's $(i = 1, ..., s)$ are the control points[14]. Each W-potential $\mathbf{U}_{p_i}$ is computed using the simple definition of Subsection 3.2.1.

The heuristic potential $\mathbf{U}_{heur}$ used in our experiments is defined as the sum of the simple W-potentials computed at a few control points.

We applied the algorithm to 3-DOF mobile robot problems. An example of generated path for a triangular robot is shown in Figure 18. This example was run using 15 control points

[13] We developed the valley-guided motion technique before the three other techniques presented in this paper. It has been implemented on a SUN III workstation, and the experiments described below were carried out on this machine.

[14] The expression defining $\mathbf{U}(\mathbf{q})$ is a discrete approximation of $\sigma \log(\int_A \exp(\mathbf{U}_p(X(p, \mathbf{q}))/\sigma)dp)$, which decreases with the distance separating the robot and the obstacles and tends toward infinity when this distance tends toward 0. The expression defining $\mathbf{U}(\mathbf{q})$ also uniformly converges toward the simpler expression $\sup_{i \in [1,s]} \mathbf{U}_p(X(p, \mathbf{q}))$ when $\sigma \to 0$. The latter expression, however, is highly degenerated for manipulator arms and produces flat valleys which are difficult to track. Therefore, it cannot be used reliably by the valley-guided planning technique.

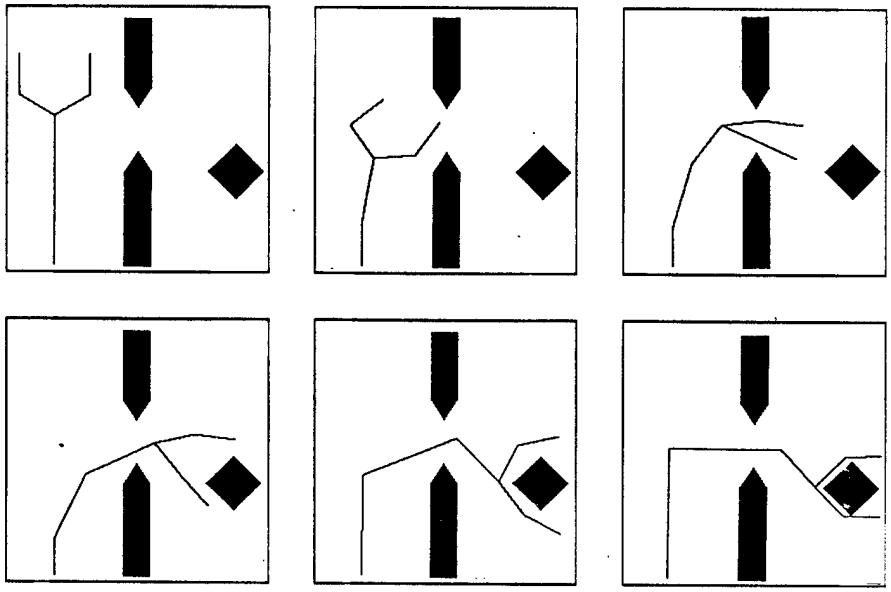


Figure 19: Path Generated for the 10-DOF Manipulator

equally distributed in the triangle boundary. We ran the algorithm with and without the heuristic potential $\mathbf{U}_{heur}$. When used, $\mathbf{U}_{heur}$ was computed by considering a single control point. In both cases, the program found a solution path. However, the heuristic potential considerably reduced the overall computational time. In the example of Figure 18, the running time was reduced from 2 minutes to less than 30 seconds.

We also applied the algorithm to the 10-DOF robot shown in Figure 14. Figure 19 shows a path generated by the planner. In this example, the C-potential $\mathbf{U}$ was computed using 70 control points equally distributed in the robot and the heuristic potential $\mathbf{U}_{heur}$ using two control points located at the end-points of the two kinematic chains. The use of the heuristic potential was necessary to obtain a path in a reasonable amount of time. (In a 10D configuration space, each point has 200 2-neighbors. The computation of the next point along a valley takes a few seconds.) The path was generated in less than 1 hour.

Notice that the path planning problem shown in Figure 19 is significantly simpler than the problem shown in Figure 15. There is more space for the robot to "maneuver". (In Figure 15, the hole in the wall is narrower and the square object is closer from the wall.) The above algorithm failed to solve the problem of Figure 15. The search was stopped after a few hours.

The above planning technique is slower than the best-first and random motion techniques for robots with few DOF's. It is also slower and definitely less reliable than the random motion technique for robots with many DOF's. The failures of the algorithm may have several causes: the search of the valley roadmap may require exponential-time computation, the valley roadmap may imperfectly represent the connectivity of the free space, it may be locally degenerated (i.e., may not be 1D), the 2-neighborhood may be insufficient to reliably

track the valleys, etc... We think that these causes might be partly remedied by investigating the concept of valley further (see [Barraquand, Langlois and Latombe, 1989]).

Notice that using the concept of valley might also be useful to improve the random motion technique. Indeed, when a gradient motion reaches a local minimum, that algorithm executes a series of non-informed Brownian motions. Instead, valley directions could be extracted around the encountered minimum and the probability laws governing the random motions could be adapted so that these motions tend to stay closer than pure Brownian motions from the valley directions.

7 Constrained Motion Technique

7.1 · Description

The constrained motion technique rests on a combination of ideas already present in the previous three techniques. On the one hand, like the best-first and random motion techniques, it makes use of a C-potential which is globally minimal (null) in the subset $\mathcal{C}_{goal}$ of goal configurations and, except when it tries to escape a local minimum, it follows the negated gradient of this C-potential. On the other hand, it uses a concept similar to the notion of valley for escaping encountered local minima.

Starting from the given initial configuration, the algorithm follows the flow of the negated gradient field of the C-potential $\mathbf{U}$ until a local minimum, $\mathbf{q}_{loc}$, is attained (gradient motion). This minimum may be a configuration where the gradient of $\mathbf{U}$ is zero or a configuration located in the boundary of $\mathcal{GC}_{free}$ (see Subsection 4.1). If $\mathbf{q}_{loc} \in \mathcal{C}_{goal}$, the problem is solved. Otherwise, the planner executes a series of **constrained motions**. A constrained motion, denoted by $M_i^+(\mathbf{q}_{loc})$ (resp. $M_i^-(\mathbf{q}_{loc})$), with $i \in [1, n]$, consists of iteratively forcing the ith configuration space coordinate, q_i, to increase (resp. decrease) by the increment Δ_i of the grid $\mathcal{GC}$, until a saddle point of the local minimum well is reached. Since $\mathbf{q}_{loc}$ is a local minimum of the C-potential, this means that the ith DOF is required to move against the force applied by the C-potential. At each iteration of the constrained motion, the remaining $n-1$ coordinates q_j, $j \neq i$, are selected according to a best-first rule. Hence, if $(q_1, ..., q_{i-1}, q_i, q_{i+1}, ...q_n)$ is the current configuration, its successor minimizes the C-potential over the set consisting of the configuration $(q_1, ..., q_{i-1}, q_i', q_{i+1}, ..., q_n)$, where $q_i' = q_i + \Delta_i$ (if $M_i^+(\mathbf{q}_{loc})$ is being executed) or $q_i - \Delta_i$ (if $M_i^-(\mathbf{q}_{loc})$ is being executed) and its neighbors. The motion thus tracks a kind of valley in the $(n-1)$-dimensional subspace orthogonal to the q_i axis. Hopefully, it ultimately reaches a saddle point of the local minimum well. The planner recognizes such an event when the C-potential decreases again. Then, it terminates the constrained motion and executes another gradient motion.

Assume for an instant that every constrained motion attains a saddle point of the local minimum well. By interweaving gradient and constrained motions, the planner constructs a graph of local minima. Each minimum, after it has been fully expanded, has $2n$ immediate successors in the graph and is connected to each of them by the concatenation of a constrained motion and a gradient motion. Whenever the planner attains a minimum, it may check if it is a new one, by comparing its coordinates with those of previously generated minima. The search of the local minima graph may be conducted in a best-first

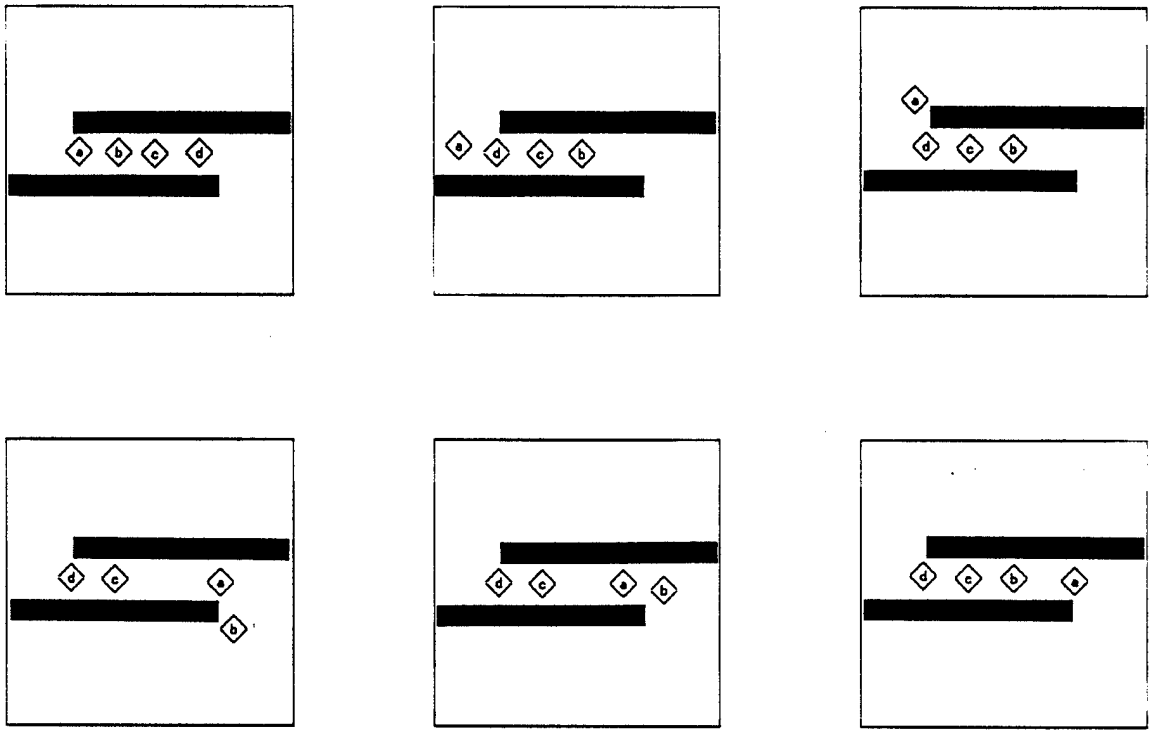


Figure 20: Coordinated Motion of 4 Mobile Robots

fashion by iteratively selecting a pending local minimum having the smallest value of the C-potential and generating its successors. In order to avoid the systematic generation of all the successors of every selected pending minimum, we have implemented a variant of this strategy. At each iteration, the implemented algorithm selects a minimum $\mathbf{q}_{loc}$ having the smallest value of $\mathbf{U}$ among those whose successors have not all been constructed yet, and it generates a new successor $\mathbf{q}'_{loc}$ of $\mathbf{q}_{loc}$. At the next iteration, if $\mathbf{U}(\mathbf{q}'_{loc}) < \mathbf{U}(\mathbf{q}_{loc})$, the algorithm naturally selects $\mathbf{q}'_{loc}$ and generates one of its successors; otherwise, it constructs another successor of $\mathbf{q}_{loc}$, if any.

However, a constrained motion $M_i^+(\mathbf{q}_{loc})$ (or $M_i^-(\mathbf{q}_{loc})$) may not terminate at a saddle point. Instead, it may hit an obstacle. In that case, the implemented algorithm merely considers that the local minimum has no successor "in the direction of" increasing (or decreasing) q_i. An alternative would be to treat the last configuration attained by the constrained motion before the collision as a successor of $\mathbf{q}_{loc}$. This successor would not be a local minimum, but it would nevertheless be included as a node of the local minima graph and then treated as any other node, except that the constrained motions executed from it should not be commanded along the q_i axis. It is rather easy to construct examples where the constrained motion technique succeeds only if these non-local-minimum configurations are inserted in the search graph.

Without further assumptions on the C-potential, the above path planning technique is not

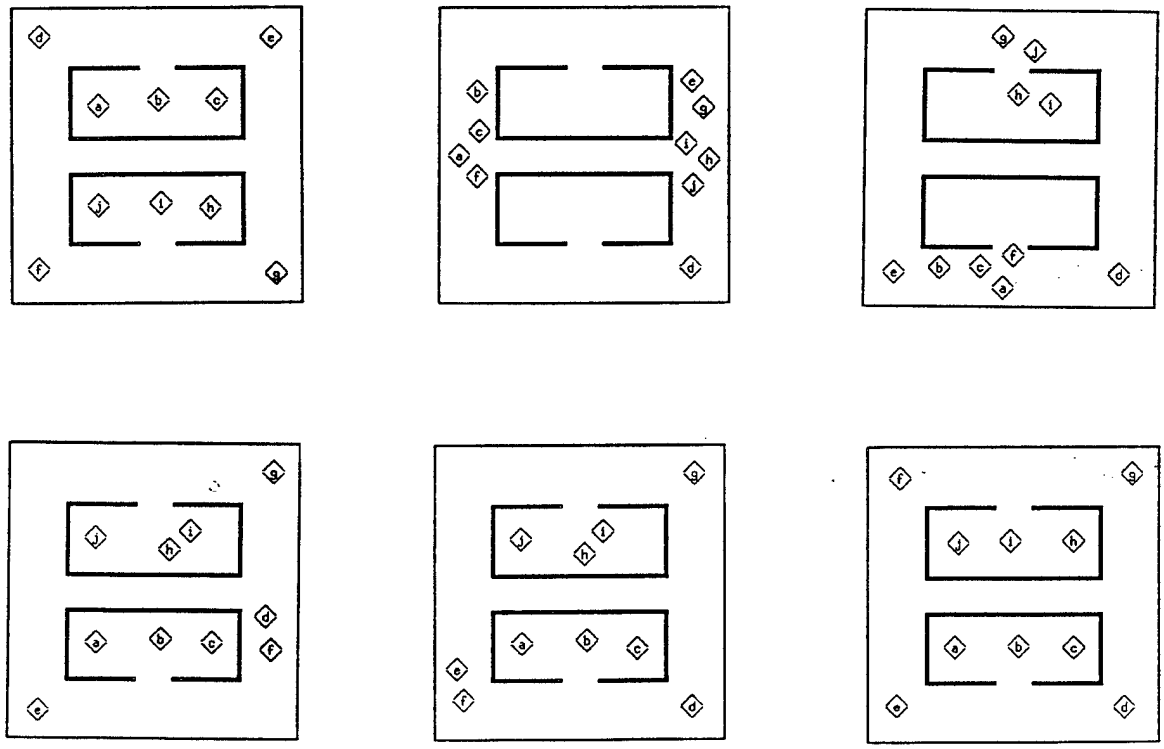


Figure 21: Coordinated Motion of 10 Mobile Robots

complete. Nevertheless, as shown below, it produces interesting experimental results.

7.2 Experimentation

We initially developed the constrained motion technique with the goal of solving planning problems involving the coordinated motions of several mobile robots in a workspace made of narrow corridors. Indeed, consider the case where a gradient motion leads two mobile robots to roll in opposite directions in a narrow corridor where they cannot pass each other. At some point, they cannot progress further toward their respective goals without colliding. Then, the gradient motion has reached a local minimum of the C-potential defined in the product configuration space of the two robots. A constrained motion corresponds to one of the robots moving backward, the other robot proceeding along the negated gradient of the C-potential, until there is sufficient room for the robots to pass each other.

We applied the technique to plan the motion of 4 robots in a workspace containing two parallel walls forming a narrow corridor (see Figure 20). Each robot is a L^1-disc of radius a denoted by $\mathcal{A}_i$, $i = 1, ..., 4$, which can only translate in the plane. Its configuration is defined as the coordinates (x_i, y_i) of the center p_i of the square in a workspace coordinate frame[15]. The configuration space of the whole robot system is a 8D space with each con-

[15]The orientation of the coordinate axes with respect to the walls has no major impact on the operations of the planner.

figuration represented by $(x_1, y_1, ..., x_4, y_4)$. In the example shown in Figure 20, the robots initially stand in the corridor made by the two walls. In the goal configuration, which is uniquely defined, the robots also stands in the corridor but in the reversed order. Figure 20 shows various intermediate configurations along the path generated by the planner. In this example, the C-potential was constructed as the sum $\sum_{i=1}^{4} \mathbf{U}_{p_i}(X(p_i, \mathbf{q}))$ of the simple W-potentials computed at the centers of the robots. The collision-checking technique of Subsection 4.1 reduces to the comparison of the L^1 distance $d_1(X(p_i, \mathbf{q}))$ to the L^1 radius a. In the same fashion, the collisions between two robots $\mathcal{A}_i$ and $\mathcal{A}_j$ were checked by comparing the L^1 distance between the points p_i and p_j and $2a$. The total planning time was 30 seconds.

Figure 21 illustrates another example on which we experimented with the planning technique. Ten mobile robots, identical to the robots of the previous example, operate in a workspace consisting of two rooms connected by corridors. No two robots can pass each other in any of the corridors, except in the lateral corridors at the four corners, the two intersections with the central corridor, and the locations facing the two doors. The whole robot system has a 20D configuration space. The initial configuration is shown in the upper lefthand corner of the figure and the goal configuration in the lower righthand corner. The figure displays several intermediate configurations along the path generated by the planner. We first ran the planner using the C-potential $\sum_{i=1}^{10} \mathbf{U}_{p_i}(X(p_i, \mathbf{q}))$, as in the previous example. The planner was able to construct a path, but this path traversed several tens of local minima. Many of these minima were due to the fact that the robots were moving in a de-organized fashion leading to intricate traffic jams. Then, we ran the planner using a slightly different C-potential obtained by adding to the previous C-potential a term that applies to each robot a force perpendicular to its current direction of motion and pointing towards its right. This additional field, which is analogous to a "magnetic field", essentially reproduces the "drive on the right" traffic rule. In our experiments, it led the various robots to move in a more organized fashion. Using this C-potential, the planner solved the problem shown in Figure 21 by generating a path traversing only a few local minima. The total path was generated in about 30 seconds, which is quite satisfactory given the high number of robots.

Since multi-robot path planning problems typically involves many DOF's, traditional global path planning methods are often impractical. More specific approaches known as "prioritizing" and "velocity tuning" have been proposed for such problems. The prioritizing approach [Erdmann and Lozano-Pérez, 1986] consists of considering the robots in sequence. When the path of a robot has been constructed, this robot becomes a mobile obstacle for the robots whose motions have not yet been planned. The velocity tuning approach [Kant and Zucker, 1986] consists of first planning the motion of each robot as if it was the only robot in the workspace and then tuning the velocities so that no two robots collide. Hence, both approaches first break the planning problem for q robots into q simpler planning problems, each involving a single robot, and then consider the interactions among their solutions. These approaches are unable to solve multi-robot problems involving tight interactions among the robots, such as the two problems shown above.

Although the constrained motion technique seems particularly suitable for planning coordi-

nated motions of multiple mobile robots, we have also experimented with it using the 8-DOF manipulator arm of Figure 12 with the same C-potential as in the experiment reported in Subsection 5.3. In the example of Figure 13, a solution path was constructed in about 3 minutes, which is comparable to the result obtained with the random motion technique. However, our experimentation with multi-joint manipulator arms has been limited, and the reliability of the constrained motion technique remains to be verified for such robots.

8 Conclusion

In this paper, we presented a collection of numerical potential field techniques for robot path planning. All these techniques apply the same general approach. They construct a potential field over the robot's configuration space, build a graph connecting the local minima of this potential, and search this graph. The graph is built incrementally and searched as it is built. The only precomputation step is aimed at the construction of the local-minimum-free W-potentials for the various control points in the robot. This step could be avoided, but it is an important one, since it allows us to construct "good" C-potentials. In addition, this precomputation occurs in a space of fixed and small dimension $d = 2$ or 3. In virtually all practical cases, the size of the graph of the local minima is reasonably small, resulting in efficient path searching.

We proposed four techniques for constructing the local minima graph. These techniques are based on different ways of escaping the encountered local minima of the C-potential. The best-first motion technique is very simple and gives excellent results for robots with few DOF's. However, it is impractical for robots with many DOF's. The random motion technique gives very good results for robots with few and many DOF's. In addition, it is highly parallelizable. The valley-guided motion technique gave significantly inferior results. However, it embeds ideas that one might improve in the future by exploring the concept of valleys more thoroughly. The constrained motion technique, although incomplete, is particularly well-adapted to solve planning problems involving the coordinated motions of several robots. It gave very good experimental results, specially for planning the coordinated motions of several simple mobile robots. The principles underlying these four planning techniques are somewhat independent of the representation of the robot's workspace and configuration space. However, their efficient implementation was made possible by the use of hierarchical bitmap representations. The planner implementing these techniques was able to solve path planning problems, whose complexity measured by the number of DOF's or the number of obstacles is far beyond the capabilities of previously implemented planners. For simpler problems, our planner is several orders of magnitude faster than most previous planners.

We feel that the techniques presented in this paper and the experimental results obtained with them make it possible to realistically envision the development of "real-time" path planners. By "real-time", we mean that the planners would be able to produce paths in a very small amount of time (say, a fraction of a second) in almost all practical situations. The construction of such a real-time planner will probably require the use of some dedicated hardware and parallel computing architecture, leading to the notion of "motion engine" just like there exist "geometric engines" for performing graphic operations (e.g., hidden line

removal). We believe that the availability of a real-time motion engine would open new perspectives on some important issues in industrial robot programming and autonomous robot control, and enable the construction of efficient robot systems operating in partially known and dynamically changing workspaces. Among the four techniques presented in this paper, the random motion technique seems to provide the best combination of qualities (time efficiency, generality, reliability). Since it is also highly parallelizable, it is probably the best candidate for developing such a motion engine. The random motion technique described in the paper makes use of un-informed probability distribution laws. If a very fast implementation of the technique was available, we could envision the inclusion of a learning module. In a specific application domain, this module would collect various pertinent statistics, for instance statistics about the distribution and the radii of the minimum wells, and refine the probability laws used by the planner, leading to even faster and more stable response times.

References

A.V. Aho, J.E. Hopcroft and J.D. Ullman (1983) *Data Structures and Algorithms*, Addison-Wesley, Reading, MA.

R.F. Anderson and S. Orey (1976) "Small Random Perturbations of Dynamical Systems with Reflecting Boundary," *Nagoya Mathematics Journal*, 60, 189-216.

V.I. Arnold (1978) *Mathematical Methods of Classical Mechanics*, Springer-Verlag, New York.

J. Barraquand, B. Langlois and J.C. Latombe (1989) "Robot Motion Planning with Many Degrees of Freedom and Dynamic Constraints," *Fifth International Symposium of Robotics Research*, Tokyo, Japan.

J. Barraquand and J.C. Latombe (1989) *Robot Motion Planning: A Distributed Representation Approach*, Report No. STAN-CS-89-1257, Department of Computer Science, Stanford University, CA.

R.A.Brooks and T.Lozano-Pérez (1983) "A Subdivision Algorithm in Configuration Space for Find-Path with Rotation," *Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, FRG, 799-806.

J.F. Canny (1987) *The Complexity of Robot Motion Planning*, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA.

V. Cerny (1985) "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *Journal of Optimization Theory and Applications*, 45(1).

B.R. Donald (1984) *Motion Planning with Six Degrees of Freedom*, Technical Report 791, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

M. Erdmann and T. Lozano-Pérez (1986) *On Multiple Moving Objects*, A.I.Memo No. 883, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

B. Faverjon (1986) "Object Level Programming Using an Octree in Configuration Space," *IEEE International Conference on Robotics and Automation*, San Francisco, CA.

B. Faverjon and P. Tournassoud (1987) "A Local Based Approach for Path Planning of Manipulators with a High Number of Degrees of Freedom," *IEEE International Conference on Automation and*

Robotics, Raleigh, NC, 1152-1159.

D. Geman and S. Geman (1984) "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6, 721-741.

R.A. Jarvis and J.C. Byrne (1987) "An Automated Guided Vehicle with Map Building and Path Finding Capabilities," *Fourth International Symposium of Robotics Research*, Santa-Cruz, CA.

K. Kant and S.W. Zucker (1986) "Toward Efficient Trajectory Planning: Path Velocity Decomposition," *International Journal of Robotics Research*, 5, 72-89.

O. Khatib (1986) "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research*, 5(1), 90-98.

S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi (1983) "Optimization by Simulated Annealing," *Science*, 220, 671-680.

D.T. Lee and R.L. Drysdale (1981) "Generalization of Voronoi Diagrams in the Plane," *SIAM Journal of Computing*, 10, 73-87.

D. Leven and M. Sharir (1987) "Intersection and Proximity Problems and Voronoi Diagrams," in [Schwartz and Yap, 1987], 187-228.

N.J. Nilsson (1980) *Principles of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA.

C. Ó'Dúnlaing, M. Sharir and C.K. Yap (1983) "Retraction: A New Approach to Motion Planning," *Proceedings of the 15th ACM Symposium on the Theory of Computing*, Boston, 207-220.

A. Papoulis (1965) *Probability, Random Variables, and Stochastic Processes*, Mc. Graw-Hill.

E. Rimon and D.E. Koditschek (1989) "The Construction of Analytic Diffeomorphisms for Exact Robot Navigation on Star Worlds," *IEEE International Conference on Robotics and Automation*, Scottsdale, AZ, 21-26.

S.A. Schneider (1989) *Experiments in the Strategic and Dynamic Control of Cooperating Manipulators*, Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, Stanford, CA.

J.T. Schwartz and M. Sharir (1983) "On the 'Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds", *Advances in Applied Mathematics*, Academic Press 4, 298-351; also in [Schwartz, Sharir and Hopcroft, 1987].

J.T.Schwartz, M.Sharir and J.Hopcroft (1987) *Planning, Geometry, and Complexity of Robot Motion*, Ablex, Norwood, NJ.

J.T.Schwartz and C.K.Yap (1987) *Algorithmic and Geometric Aspects of Robotics*, Lawrence Erlbaum Associates, Hillsdale, NJ.

J. Serra (1982) *Image Analysis and Mathematical Morphology*, Academic Press.